Technická univerzita v Košiciach
Fakulta elektrotechniky a informatiky

Generovanie náhodných čísel na platforme Windows

Bakalárska práca

2021 **Marek Rohač**

Technická univerzita v Košiciach
Fakulta elektrotechniky a informatiky

Generovanie náhodných čísel na platforme Windows

Bakalárska práca

Študijný program:	Počítačové siete
Študijný odbor:	Informatika
Školiace pracovisko:	Katedra elektrotechniky a multimediálnych telekomunikácií (KEMT)
Školiteľ:	prof. Ing. Miloš Drutarovský, CSc.
Konzultant:	

Košice 2021 **Marek Rohač**

Abstrakt v SJ

Cieľom teoretickej časti tejto práce je oboznámiť čitateľa s problematikou generovania náhodných čísel v oblasti informačnej bezpečnosti. Následne pomocou opisu štruktúr a mechanizmov operačného systému Microsoft Windows vysvetliť proces generovania náhodných čísel na tejto platforme. Praktická časť implementuje, pomocou programovacieho jazyka C, knižničné funkcie a predprogramované systémové rozhrania na produkciu náhodných výstupov. Kvalita takto vzniknutých dát je následne overená pomocou sád štatistických testov. Pomocou experimentu v prostredí virtuálneho stroja s rovnakým operačným systémom je overená bezpečnostná chyba súvisiaca s problematikou tejto práce.

Kľúčové slová v SJ

CAPI, CNG, CSPRNG, HRNG, language C, OpenSSL, Windows, PRNG, RNG, TRNG, winapi

Abstrakt v AJ

The aim of the theoretical part of this work is to acquaint the reader with the issue of generating random numbers in information security. Subsequently, by describing the structures and mechanisms of the Microsoft Windows operating system, we will explain the generation process on this platform. In the practical part are implemented library function and programmed system interfaces for the production of random outputs by C programing language. The quality of the data generated in this way is then verified by using statistical test suits. By experiment in enviroment of virtual machines with OS Windows is checked security problem related to the topic of this work.

Kľúčové slová v AJ

CAPI, CNG, CSPRNG, HRNG, language C, OpenSSL, Windows, PRNG, RNG, TRNG, winapi

Bibliografická citácia

ROHAČ, Marek. *Generovanie náhodných čísel na platforme Windows*. Košice: Technická univerzita v Košiciach, Fakulta elektrotechniky a informatiky, 2021. 67s. Vedúci práce: prof. Ing. Miloš Drutarovský, CSc.

61344

TECHNICKÁ UNIVERZITA V KOŠICIACH

FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Katedra elektroniky a multimediálnych telekomunikácií

ZADANIE BAKALÁRSKEJ PRÁCE

Študijný odbor: **Informatika**

Študijný program: **Počítačové siete**

Názov práce:

Generovanie náhodných čísel na platforme Windows

Random Numbers Generation on Windows Platform

Študent: **Marek Rohač**

Školiteľ: **prof. Ing. Miloš Drutarovský, CSc.**

Školiace pracovisko: **Katedra elektroniky a multimediálnych telekomunikácií**

Konzultant práce:

Pracovisko konzultanta:

Pokyny na vypracovanie bakalárskej práce:

Na základe dostupných informácií naštudujte a opíšte mechanizmy generovania náhodných čísel, ktoré sú dostupné pre aplikačné programy na platforme Windows. S využitím jazyka C a dostupných API rozhraní otestujte základné parametre generovania náhodných čísel so zameraním na kryptografické aplikácie. Opíšte a experimentálne overte možnosť útoku na generátor náhodných čísel vo vybraných virtualizačných nástrojoch, ktoré je možné využiť na hosťovskom počítači s operačným systémom Windows.

Jazyk, v ktorom sa práca vypracuje: slovenský

Termín pre odovzdanie práce: 28.05.2021

Dátum zadania bakalárskej práce: 30.10.2020

.....

prof. Ing. Liberios Vokorokos, PhD.

dekan fakulty

Čestné vyhlásenie

Vyhlasujem, že som záverečnú prácu vypracoval samostatne s použitím uvedenej odbornej literatúry.

Košice, 4. 5. 2021 .....

Vlastnoručný podpis

Poďakovanie

Na tomto mieste by som rád poďakoval svojmu vedúcemu práce *prof. Ing. Milošovi Drutarovskému, CSc.* za jeho čas a odborné vedenie počas riešenia mojej záverečnej práce.

Obsah

Zoznam obrázkov

Zoznam tabuliek

Zoznam zdrojových kódov

Zoznam skratiek

AES Advanced Encrytion Standard.

ANC Average Number of Cycles.

API Application Programming Interface.

BBS Blum Blum Shub generator.

CAPI Cryptographic Application Programming Interface.

CNG Cryptography API: Next Generation.

CPU Central Processing Unit.

CryptoAPI Cryptographic Application Programming Interface.

CSP Cryptographic Service Provider.

CSPRNG Crypographically Secure Random Number Generator.

CV Critical Value.

DRNG Deterministic Random Number Generator.

GCC GNU Compiler Collection.

HRNG Hardware Random Number Generator.

LFSR Linear-Feadback Shift Register.

n-DRNG non-Deterministic Random Number Generator.

OS Operating System.

PRNG Pseudo-Random Number Generator.

QPC Query Performance Counter.

QPF Query Performance Frequency.

RBG Random Bit Generator.

RC Rivest Cipher.

RNG Random Number Generator.

SHA Secure Hash Algorithm.

TPM Trusted Platform Module.

TRNG True Random Number Generator.

TSC Time Stamp Count.

UEFI Unified Extensible Firmware Interface.

VM Virtual Machine.

Úvod

Náhodné čísla sú nevyhnutnou súčasťou každodenného použitia počítača. Ich využitie pri práci operačného systému je rozsiahle a prebieha takmer neustále. Podieľajú sa pri zabezpečení sieťovej, a aj počítačovej ochrany pri používaní zariadenia. Poskytujú teda ochranu pred možnými útokmi, prostredníctvom ich využitia v rôznych kryptografických algoritmoch. Kvalita náhodných dát, vygenerovaných počítačom, teda odzrkadľuje bezpečnosť zvolenej platformy, ktorá je poskytnutá používateľovi.

Cieľom tejto práce je definovať a klasifikovať prostriedky na tvorbu náhodne vygenerovaných čísel. Následne opisom špecifikujeme vývoj kryptografických rozhraní od svojho vzniku až po súčasnosť operačného systému Windows. Vysvetlíme si princípy testovania kvality náhodných čísel. Možnosti produkcie týchto dát v uvedenom prostredí špecifikujeme. Na rozhraniach a knižničných funkciách vykonáme experimenty. Zvolené metódy merania a testovania charakterizujeme. V opise uvedieme aj vplyvy možných chýb merania. Experimenty spojené s časovým testovaním sú vykonané na troch rôznych konfiguráciách, s rovnakým operačným systémom. Dôvodom na použitie viacerých zariadení je kvalitnejšia interpretácia výsledkov. Ďalším krokom je generovanie náhodných dát pomocou vybraných rozhraní. Následné takto vzniknuté údaje otestujeme štatistickými testami. Na základe výstupov z testovanie zistíme kvalitu našich dát. Súčasťou práce je aj bezpečnostný problém v prostredí virtuálneho stroja s operačným systémom Windows. Špecifikáciu rizika vykonáme prostredníctvom opisu a experimentu. Cieľom je praktické overenie aktuálnosti bezpečnostnej chyby, ktorá vzniká pri generovaní náhodných čísel tesne po obnovení snímky obrazu systému.

Vyhodnotenie výsledkov, ktoré je získané vyššie uvedenými praktikami, je obsahom samostatnej kapitoly tejto práce. Opísaná metodika je uskutočnená na celosvetovo najpoužívanejšom operačnom systéme spoločnosti Microsoft – Windows 10, v 64-bitovej verzii. Na tvorbu programov je použitý programovací jazyk C so 64-bitovým GCC prekladačom.

1 Typy generátorov náhodných čísel

Pod pojmom generovanie náhodných čísel vzniká predstava jednoduchého procesu ako napríklad hod kockou. V sfére počítačov sa pri tomto deji používajú špeciálne nástroje – *generátory* [1], ďalej RNG[1]. Výstupnými hodnotami sú sekvencie bitov. Ich neskoršia grafická reprezentácia v počítači môže byť číselná alebo vo forme znakov ASCII tabuľky, prislúchajúcich vygenerovaným postupnostiam. Dôsledkom toho sa čitateľ môže v problematike RNG frekventovane stretnúť s pojmom – *generátor náhodných bitov* RBG[2]. V oblasti informačnej bezpečnosti je ich použitie skutočne rozsiahle. Radíme ich preto do množiny tzv. *kryptografických primitív*. V kryptografii tento pojem zahŕňa algoritmy, respektíve nástroje, ktoré tvoria základ pre správne fungovanie kryptografických funkcií. Podľa dokumentu Národného Inštitútu pre Štandardy a Technológie (ďalej NIST) [2], ich delíme podľa spôsobu tvorby výstupných hodnôt na:

- deterministické – *ang. deterministic RNG* (DRNG),
- ne-deterministické – *ang. non-deterministic RNG* (n-DRNG).

V tejto práci použijeme spomenuté delenie za hierarchicky najvyššie a ďalej opísané v nasledujúcich podkapitolách.

1.1 Ne-Deterministické RNG

Proces generovania náhodných čísel prebieha pri tomto type generátora na najnižších úrovniach počítača – **hardvére**. Výstup týchto zariadení je závislý od jedného alebo viacerých fyzikálnych dejov, ktoré musia byť štatisticky nepredvídateľné. Celkovo môžeme tieto javy rozdeliť na fyzikálne procesy:

- s kvantovou náhodnosťou,
- bez kvantovej náhodnosti.

[1] Z ang. Random Number Generator.
[2] Z ang. Random Bits Generator

Dôvodom použitia práve tohto odvetia fyziky je ideálna vlastnosť pre generovanie – aktuálna nemožnosť, resp. neschopnosť predikcie týchto procesov. Za spomenutie tiež stojí, že rýchlosť generovania čísel je v tomto prípade pomalšia ako pri deterministických generátoroch. Aj dôsledkom toho sa stretávame s menším zastúpením týchto generátorov v bežnej prevádzke. Veľmi dobrým príkladom použitia je vojenské odvetvie. Najmä pri šifrovaní a následnej preprave veľmi dôležitých, resp. citlivých správ prostredníctvom počítačových sieti. Na označenie takýchto generátorov sa používa skratka – TRNG[3].

1.1.1 Fyzikálne procesy s kvantovou náhodnosťou

Základným zdrojom náhodnosti sú v tomto prípade mechanizmy kvantovej fyziky na atómovej a subatómovej úrovni. Príkladom takýchto dejov môže byť:

- rozpad jadra rádioaktívnych prvkov,
- výstrelový šum – *z ang. Shot/Poisson noise*[3],
- prechod fotónov cez polo-priesvitné zrkadlo[3][4],
- a iné.

Na overenie skutočnej náhodnosti dát sa v tomto prípade používajú tzv. **Bellove testy** [5]. Za spomenutie určite stojí aj tzv. **Geigerovo počítadlo** (*z ang. Geiger counter*). Tento prístroj sa požíva na zisťovanie ionizujúceho žiarenia atómov. Pri pripojení k počítaču slúži tiež ako zdroj náhodných dát.

1.1.2 Fyzikálne procesy bez kvantovej náhodnosti

Základ týchto dejov tvoria tepelné procesy. Detekcia je v tomto prípade jednoduchšia ako pri kvantových javoch. Uvedené deje sú náchylnejšie na útok, pri ktorom sa zníži prevádzková teplota daného systému na nižšiu ako, pri ktorej je zariadenie na tvorbu náhodných dát, schopné pracovať správne. Príkladom tepelných dejov je **tepelný šum**, ktorý produkuje rezistor. Ten je zosilnený tak, aby poskytoval náhodný zdroj napätia. Okrem spomenutého sa používa aj **atmosferický** [6] a **lavínový šum**(*z ang. Avalanche noise*) [7]. Veľmi jednoduchým a nepredvídateľným dejom je tiež tzv. **posun hodín**[4] opísaný v [8].

[3] Z ang. True Random Number Generator
[4] Z ang. Clock drift

1.2 Deterministické RNG

DRNG[5], resp. DRBG[6], je taký generátor náhodných čísel/bitov, ktorý potrebuje na vygenerovanie výstupu prístup k zdroju náhodnosti, minimálne pri spustení – *inicializácií*. Následne generátor aplikuje pripravený algoritmus. Výstupom je postupnosť bitov, vytvorená na základe tajnej počiatočnej hodnoty, z ang. **seed**. Tento pojem charakterizujeme ako reťazec bitov, ktorý sa používa pri inicializácií kryptografických nástrojov.

Z vyššie uvedeného vyplýva, že tento typ generátorov produkuje **pseudo-náhodné**, teda nie úplne náhodne dáta. Kvôli tomu sa takéto generátory často označujú ako PRNG (z ang. Pseudo-Random Number Generator) [9]. V uvedenom dokumente sa obdobne nachádzajú odporúčania a štandardizačný popis použitých mechanizmov, stavov a funkcií pre deterministické generátory. Ďalším, ale nie menej podstatným faktom ostáva, že celý proces generovania dát je podmienený počiatočnou hodnotou **seed**. Tento údaj predstavuje najzraniteľnejšie miesto týchto generátorov. Ak sa útočníkovi podarí získať túto bitovú sekvenciu, tak na základe predpísaného postupu dokáže presne zreprodukovať rovnakú výslednú postupnosť bitov.

1.2.1 Pseudo-náhodné a kryptografický bezpečné RNG

V súčasnosti existuje veľa voľne dostupných internetových zdrojov kde sú zverejnené zoznamy pseudo-náhodných generátorov spoločne s ich kladmi a zápormi daného typu. Preto v tejto práci nebudeme pokračovať v tomto smere. Vývoj v oblasti výpočtovej techniky a technológií samotných, však v priebehu poslednej dekády rapídne pokročil. Aj dôsledkom toho je nutnosť mať k dispozícií generátory, ktorých kvalita výstupu nie je ľahko napadnuteľná, respektíve prelomiteľná. Na základe tejto myšlienky a dokumentu [10], následne delíme PRNG na:

- bežné PRNG,
- kryptograficky bezpečné PRNG – CSPRNG[7].

Hlavným cieľom CSPRNG je vyprodukovať výstupné dáta, ktoré sú kvalitou takmer na nerozoznanie od TRNG. Kryptograficky bezpečné generátory musia spĺňať 3 základne podmienky:

[5] Z ang. Deterministic Random Number Generator
[6] Z ang. Deterministic Random Bit Generator
[7] Z ang. Cryptographically Secure PRNG

- úspešné absolvovanie štatistických testov,
- úspešný The next bit test [11],
- odolnosť voči narušeniu stavu – z ang. state compromise extensions.

Pod posledným pojmom sa rozumie prípad, v ktorom útočník zistí ľubovoľný zo stavov daného generátora. Napriek tejto znalosti nebude možné zistiť predchádzajúci ani budúci stav a teda zostaviť pôvodné dáta. V tejto súvislosti sa zvyknú používať termíny tzv. *„Spätná/Do-predná bezpečnosť"*, ktoré lepšie popisujú danú bezpečnostnú podmienku.

Spätná bezpečnosť (z ang. Backward Security, tiež známa ako *break-in recovery*), popisuje zabezpečenie pre prípad zistenia ľubovoľného stavu v určitom čase. Následne však nie je možné určiť akýkoľvek budúci výstup. Tento problému rieši dostatočne silný zdroj náhodnosti. Údaje v jednotlivých stavoch sa následne periodicky menia, resp. aktualizujú pomocou tejto náhodnosti.

Druhý prípad, teda **do-predná bezpečnosť**[8], na druhej strane hovorí o bezpečnosti pri zisťovaní stavu generátora smerom k zdrojovým dátam. Zjednodušene povedané, ak útočník zistí niektorý zo stavov generátora, tak aj napriek tejto znalosti nebude možné zistiť predchádzajúci stav a zároveň aj dáta. Tento prvok bezpečnosti sa dá ľahko zaručiť tým, že funkcia, ktorá posúva stav dopredu, bude jednosmerná. Príkladom sú hašovacie funkcie.

Označenie „kryptograficky bezpečný RNG" aj napriek splneniu uvedených podmienok nie je veľmi jednoduché získať. Dôsledkom toho v praxi trvá roky testovania, kým certifikačná organizácia označí algoritmus ako CSPRNG.

Uplatnenie kryptograficky bezpečných RNG je konkrétne v oblasti kryptografickej bezpečnosti. Takéto generátory sa uplatňujú pri generovaní:

- kľúčov pre kryptografické algoritmy – šifrovanie/dešifrovanie,
- jednorázových čísel – z ang. nonce [12],
- dodatočných vstupov – z ang. salt [13].

Bezpečnosť kryptografického systému je hodnotená práve na základe veľkosti náhodne vygenerovaných kľúčov. V súčasnosti dizajn väčšiny CSPRNG vieme rozdeliť do troch kategórií. Jedným z nich sú kryptografické primitíva. Konkrétne tieto pracujú na princípe hashovacích funkcií, blokových a prúdových šifier, ktoré sú vysvetlené v dokumente [14, kap. 5]. Ďalší typ návrhu je založený na základe zložitých matematických problémov. Hádam najznámejším je tzv. Blum Blum

[8] Z ang. Forward Security.

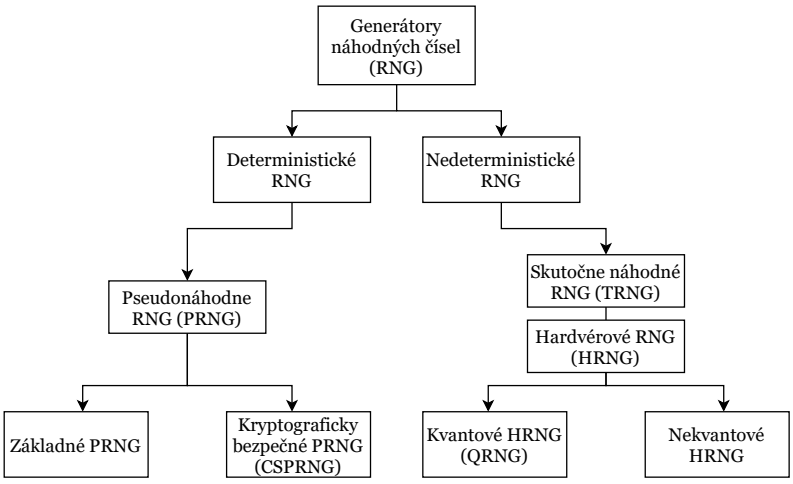
Shub generátor - BBS, ktorý je predmetom opisu práce [15]. Posledný dizajn charakterizujeme slovami – **špeciálny návrh**. Do tejto kategórie zahŕňame algoritmy, ktorých dizajn bol vyvinutý, resp. navrhnutý špeciálne pre splnenie vyššie spomenutých podmienok CSPRNG. Príkladom takýchto generátorov sú:

- Fortuna – používa ho macOS aj OS Linux – [16],
- Rivest šifra – ozn. RC – [17, kap. 3.1.1],
- Pokročilý šifrovací štandard – ozn. AES[9] – [18].

Opisu niektorých zo spomenutých generátorov sa venuje aj profesor Levický v knihe *Kryptografia v informačnej bezpečnosti* – [14, kap. 5].

1.3 Súhrnná klasifikácia generátorov

Na základe vyššie uvedených skutočností je evidentné, že v tejto problematike dochádza v počítačovej sfére k častej zámene pomenovaní. Z dôvodu kvalitnejšej orientácie čitateľa v problematike je nižšie znázornená schéma číslo 1.1.

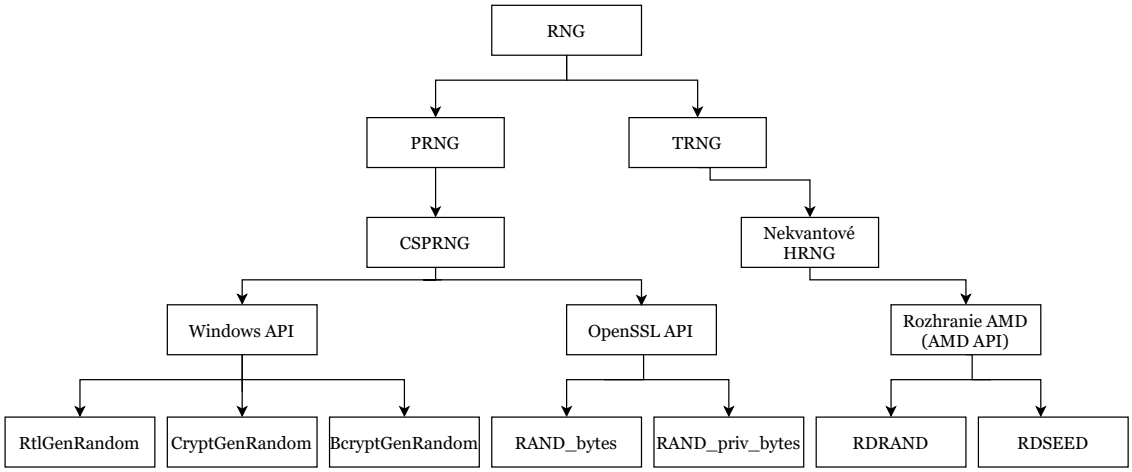


Obr. 1.1: Schéma rozdelenia RNG

V tejto práci sa zameriame na generátory, ktoré sú vhodné pre kryptografické účely. Presnejšie aplikujeme naše meracie a testovacie metódy na rozhrania operačného systému Windows, ktoré sprostredkujú tieto služby na počítači. Okrem nich opíšeme a otestujeme aj iné možnosti produkcie náhodných dát, ktoré má používateľ na tejto platforme. Vyššie uvedené podmienky spĺňajú len CSPRNG a HRNG.

[9] Z ang. Advanced Encryption Standard

Pri hardvérových generátoroch sa pri experimentoch zameriame iba na nekvantové. Dôvodom je, že dáta si chceme vygenerovať sami, bez nutnosti použitia prostriedkov tretích strán. Schéma číslo 1.2 znázorňuje RNG a rozhrania, ktoré sú predmetom skúmania tejto práce.

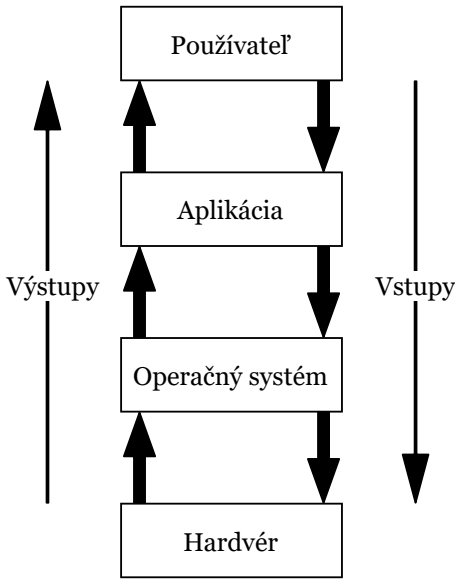


Obr. 1.2: Schéma rozhraní použitých v práci

2 Operačný systém Windows

Pred analýzou konkrétnej implementácie je nutná definícia niektorých pojmov. Prvým bude samotný OS.

Operačný systém je komplexný program – softvér, ktorý spravuje hardvér počítača. Tvorí základ pre aplikačné programy. Pracuje ako sprostredkovateľ medzi používateľom počítača a jeho hardvérom. Inými slovami, prácu tohto systému by sme v bežnej pracovnej sfére vedeli prirovnať k činnosti riaditeľa, respektíve manažéra spoločnosti. Jeho úlohou je správne a efektívne prerozdelenie práce – *činnosti aplikácií*, medzi svojich zamestnancov – *súčasti hardvéru*, na základe aktuálneho dopytu – *interakcií používateľa*.



Obr. 2.1: Úrovne počítača

Aby sa zabezpečila správna funkcionalita, tak OS musí riešiť množstvo s tým súvisiacich problémov. Ide napríklad o **plánovanie úloh**, z ang. *task sheduling*. Ďalej **správu pamäte zariadenia**, **monitorovanie hardvérového príslušenstva**, **spracovanie vstupov a výstupov z/na príslušné zariadenia** (klávesnica, myš, ...), a iné. Dôležitým aspektom pri návrhu alebo voľbe OS je závislosť od miesta jeho nasadenia. Niektoré operačné systémy sú navrhnuté so zameraním na jedno-

duché používateľské použitie. Ide o tzv. *user-friendly*, teda používateľsky prívetivé systémy. Ich použitie je bežné a príkladom môže byť samotný OS Windows 10. Ďalšiu kategóriu tvoria systémy zamerané na poskytnutie čo najlepšej efektívnosti nejakej služby. Príkladom sú serverové zariadenia. Väčšinou je však nutná vyššia odbornosť pri práci s týmito OS. V súčasnosti má už používateľ k dispozícii aj vysoko efektívne, a zároveň aj používateľsky prívetivé operačné systémy. Viac pozornosti k tejto problematike je venované v [19] a [20].

Súčasťou každého dobrého operačného systému je implementácia kryptografických modulov a ich aplikačných rozhraní (ďalej API). Ich súčasťou sú okrem iných aj funkcie, ktoré vykonávajú služby RNG. **Náhodné dáta** sú dôležitou a zároveň potrebnou súčasťou viacerých kryptografických algoritmov, respektíve systémov. Tie sú použité **na zabezpečenie** samotného **systému** a **sieťovej komunikácie** s inými zariadeniami. Bezpečnosť proti útokom je teda priamo úmerná kvalite náhodných dát a správnej implementácii kryptografických algoritmov vo vnútri týchto modulov. Okrem RNG je ďalším príkladom ich obsahu hašovacie algoritmy, ktoré majú v kryptografii taktiež široké uplatnenie. Najznámejšími a v súčasnosti najpoužívanejšími je zbierka algoritmov SHA[1]. Viac informácií k tejto problematike sa uvádza v [21].

V súvislosti s generovaním náhodných dát je potrebné definovať pojem – **náhodnosť**, respektíve **entropia**[2]. Národný inštitút štandardov a technológií NIST v dokumente [22] a [23] definuje tento pojem. Následné vety budú parafrázou definícií v uvedených publikáciách. **Entropia** je matematický pojem pre náhodnú premennú X. Jej hodnota nám určuje množstvo vopred očakávaných informácií, poskytnutých zdrojom tejto premennej. Hodnota entropie je vždy viazaná na zdroj. Jej znalosť je výsledkom pozorovania, respektíve analýzy zdroja tejto náhodnosti. Okrem uvedenej slovnej definície je definovaná aj matematická formulácia dostupná na [24]. Pre pochopenie súvislostí nám vystačí aj slovná definícia.

2.1 História kryptografických API na OS Windows

Operačný systém Windows bol prvýkrát implementovaný 27. júla v roku **1993**. Niesol označenie „NT - New Technology“. Od vzniku až po súčasnosť došlo k viacerým viditeľných grafickým zmenám. Priebežným úpravám sa nevyhla ani architektúra systému. Dôležité z hľadiska obsahu práce sú roky **1996** a **2006**.

[1] Z ang. Secure Hash algorithm
[2] Z ang. *Entropy*

2.1.1 CryptoAPI

Prvý zo spomenutých rokov priniesol novú verziu OS – Windows NT 4.0, ktorej obsahom bolo tzv. **rozhranie na programovanie kryptografických aplikácií**, z ang. *Cryptographic Application Programming Interface*. Pre jeho názov sa zaužívala skratka **CryptoAPI**, resp. **CAPI**. Oficiálny návrh použitého riešenia v rozhraní však spoločnosť Microsoft nezverejnila.

Základom kryptografického modulu sú tzv. *kryptografické primitíva*. Tento pojem trochu rozšírime. Pomenovanie **kryptografické primitíva** označuje jednotlivé nízko-úrovňové algoritmy, ktoré sa **frekventovane používajú** v počítačovej bezpečnosti pri rôznych kryptografických protokoloch alebo aplikáciach. Realizujú ich dynamické knižnice („.dll“ súbory). Tieto algoritmy úzko spolupracujú s tzv. **poskytovateľmi kryptografických služieb** (ďalej CSP). Primitíva tvoria základ pre správne fungovanie ďalších funkcií alebo programov. V tejto súvislosti uvádzame pojem **kryptografické aplikácie**. Ten predstavuje označenie pre algoritmy, ktoré zabezpečujú bezpečný prenos a utajenie údajov. Uvedené programy realizujú dva základné úkony. Na základe toho rozdeľujeme aj funkcie opisovaného CAPI, na:

- certifikačné,
- kryptografické.

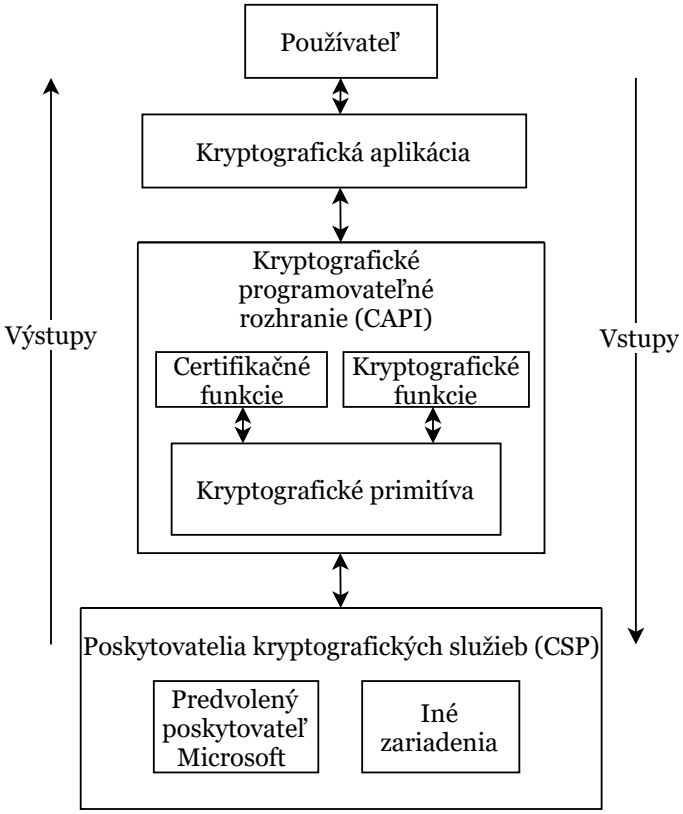
Príkladom použitia certifikačných je proces autentizácie[3] pomocou digitálneho certifikátu. V prípade druhého bodu je typické šifrovanie a dešifrovanie textu. Pod vyššie uvedeným rozdelením si môžeme predstaviť rozhrania plné funkcií. Každá z nich slúži na nejakú operáciu. Vykonávanie jednotlivých operácií je zabezpečené postupným volaním základných algoritmov – kryptografických primitív.

CryptGenRandom je funkcia z kategórie primitív a slúži na generovanie kryptograficky bezpečných náhodných dát. Výstupy sú následne použité napríklad ako zdroj tajnej hodnoty - seed. Schéma číslo 2.2 zjednodušene znázorňuje vyššie opísané skutočnosti CAPI rozhrania.

Pomocou analýz boli v minulosti odhalené rôzne chyby a zraniteľné miesta CAPI implementácie. **Nedostatky** boli zistené aj pri CSPRNG. CryptGenRandom používal na generovanie výstupu systémovú entropiu a hashovací algoritmus SHA-1´[4] v kooperácii s RC4 ([17]) šifrou. Spomenuté riešenie však neposkytuje

[3] Autentizácia – proces overenie identity.

[4] Rozdiel oproti klasickému SHA-1 bol v zmenenom poradí inicializačných vektorov



Obr. 2.2: Schéma architektúry CryptoAPI

žiadnu spätnú ani do-prednú bezpečnosť, ktorá bola opísaná pri CSPRNG. Podrobný opis použitých metód, nástrojov, útokov a dosiahnutých výsledkov analýzy tohto generátora je dostupný v dokumente [25].

2.1.2 Kryptografické rozhranie novej generácie

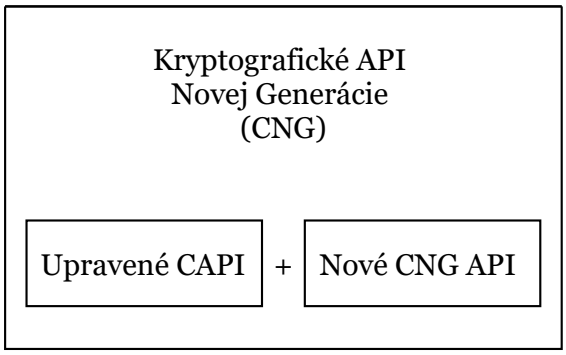
Vzhľadom na problémy návrhu kryptografického rozhrania CryptoAPI bola **nutná** jeho **inovácia**. Ta bola uverejnená v roku **2006**, spoločne s novou verziou OS – Windows Vista. Rozhranie zmenilo názov na kryptografické API novej generácie (ďalej CNG[5]). Dôležité je, že došlo k rozšíreniu a taktiež úprave pôvodného rozhrania. Dôsledkom toho sa zabezpečila aj spätná **kompatibilita** voči už nasadeným systémom. Schéma 2.3 znázorňuje uvedenú skutočnosť.

Rozhranie CNG prinieslo množstvo nových funkcií, pričom ich konštrukcia bola vytvorená na základe noriem definovaných v dokumente FIPS 140-2[26]. Príkladmi sú:

- nový spôsob konfigurácie rozhrania,
- nové kryptografické rozhranie pre jadro systému – *z ang. kernel-mode*,

[5]Z ang. Cryptography API: Next Generation

- oddelenie úložiska od operácií algoritmov,
- vylepšený proces izolácie od operácií s dlhodobým kľúčom,
- vylepšenie v oblasti uloženia, obnovenia, importu a exportu kľúčov,
- podpora kryptografie pomocou eliptických kriviek,
- podpora piatich režimov[6] v šifrovacom rozhraní, pri šifrovaní pomocou symetrických blokových šifier
- a iné.



Obr. 2.3: Schéma architektúry po inovácií

Zmeny nastali aj v oblasti RNG. V rozhraní je k dispozícií možnosť zmeniť predvolený generátor náhodných čísel. Taktiež je možné zameniť ho v CSP. Avšak táto zmena nie je možná v prípade hlavného poskytovateľa týchto služieb, ktorým je Microsoft Base CSP. Vďaka tomu je tiež možné špecifikovať určitý typ generátorov k určitému volaniu príslušnej funkcie.

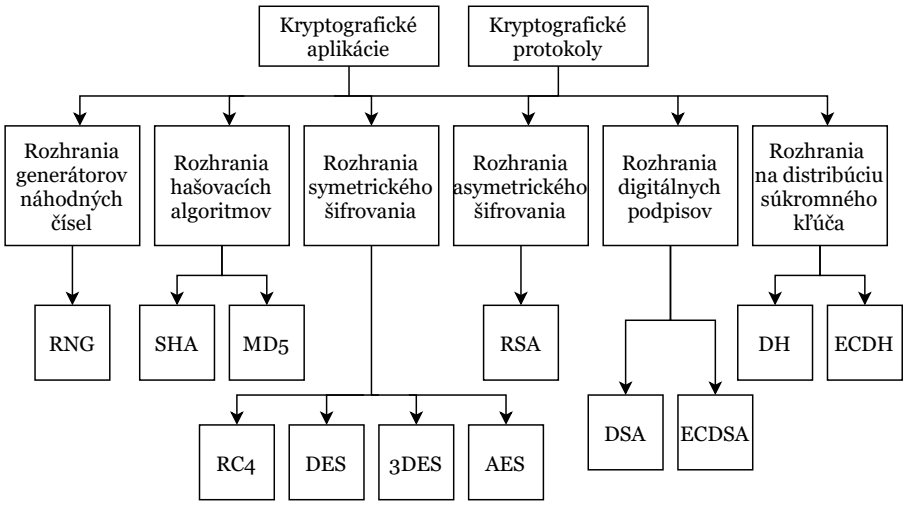
V prípade CNG rozhrania bola uverejnená aj architektúra, ale len niektorých častí. Spoločnosť v oficiálnej dokumentácii k rozhraniu zverejnila návrh i použité algoritmy. Na ich základe je vytvorená schéma 2.4.

2.2 Aktuálny prístup k rozhraniu RNG

Postupom času dochádzalo k úprave a vylepšeniam API. Dokument [27, kap. 2], opisuje prístup ku generátoru náhodných čísel v CNG module[7]. Základné mechanizmy systému boli lepšie vysvetlené práve v skorších verziách. Nasledujúce

[6] Elektronická kódovaná kniha (ECB), Zreťazené blokové šifrovanie (CBC), Šifrovanie pomocou zašifrovaného textu (CFB), CBC čítačový režim (CCM) a Galoisov čítačový řezim (GCM).

[7] Prístup bol demonštrovaný v OS Windows 8



Obr. 2.4: Schéma architektúry CNG primitív

myšlienky sú parafrázami vyššie uvedenej publikácie a dokumentu [28]. Publikácia [28] opisuje obdobne prístup k RNG, avšak v aktuálnom OS Windows 10.

Funkcie rozhrania sú prístupné v dvoch režimoch:

- režim jadra – z ang. *kernel mode*,
- používateľský režim – z ang. *user mode*.

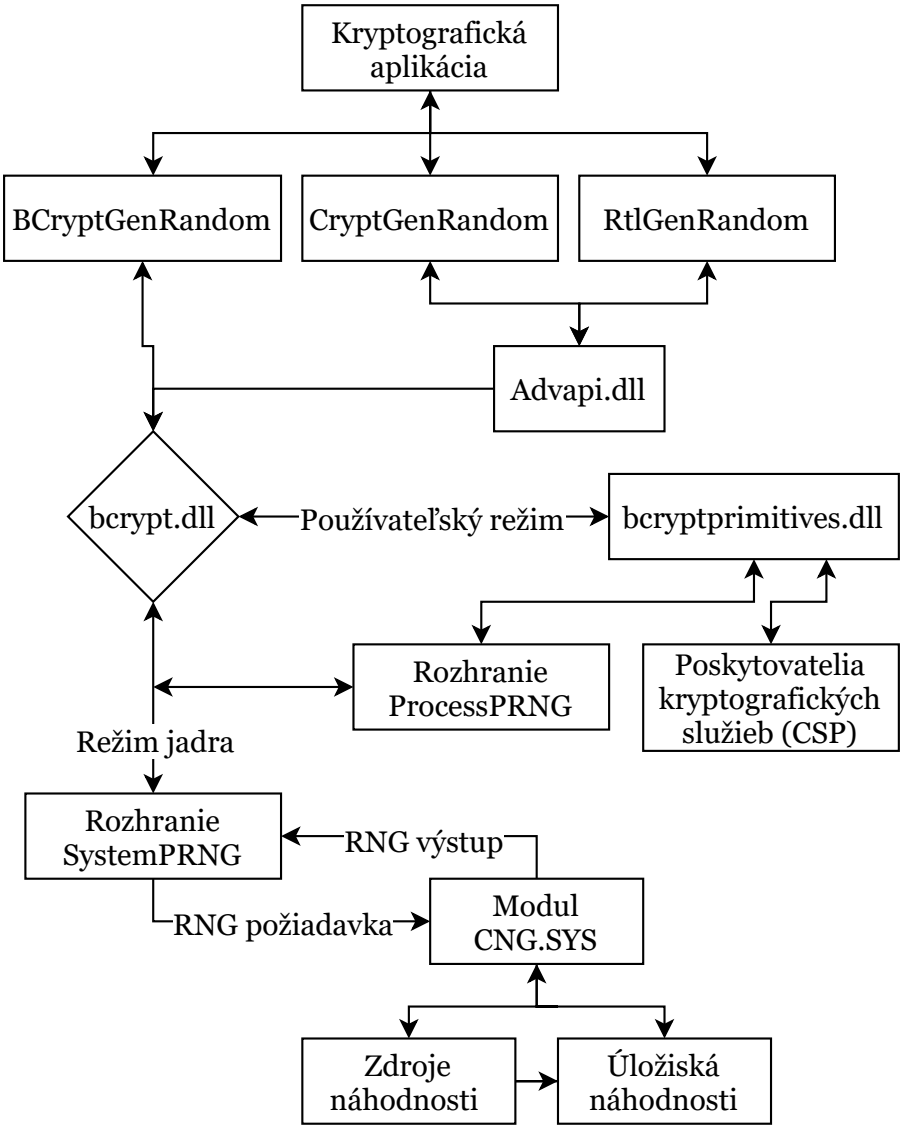
V oboch režimoch je prístup k službám rozhrania identický. Rozdiel je v spôsobe, účeloch volania a prerušeniach systému. Režim jadra je využívaný výhradne aplikáciami a komponentami jadra OS[8]. Na realizáciu tohto úkonu slúži rozhranie SystemPRNG, ktoré je súčasťou ovládača, resp. modulu - **CNG.SYS**. V prípade používateľa je prístup k generátoru zložitejší. Je riešený pomocou rozhranie ProcessPRNG. To je súčasťou dynamicky pripojenej knižnice - **bcryptprimitives.dll**. Tá následne získa prístup k softvérovému RNG a entropií pomocou SystemPRNG v module CNG.SYS. Schéma 2.5 opisuje tento prístup už aj s primárne určenými funkciami a rozhraniami, ktoré aplikácie volajú pri žiadosti o náhodné dáta.

Používateľ má možnosť zvoliť si iného poskytovateľa kryptografických služieb na získanie prístupu k softvérovému RNG. Teda OS nie je nevyhnutne odkázaný iba na prednastavené riešenie a poskytuje užívateľovi väčšiu voľnosť pri voľbe CSP.

2.3 Infraštruktúra použitých PRNG

OS Windows používa na generovanie náhodných čísel viacero typov a označení pre jednotlivé PRNG. V tejto podkapitole je uvedená ich charakteristika. In-

[8]Prístup používateľa k tomuto módu je možné realizovať implementáciou vlastného modulu



Obr. 2.5: Schéma prístupu kryptografických aplikácií k náhodným číslam

formácie boli čerpané z [28, str. 1].

Ako prvý opíšeme tzv. **základný PRNG** (z ang. *Basic PRNG*). Na všetko generovanie náhodných čísel je použitý algoritmus AES-256 v počítadlovom režime, ktorý bol spomenutý v prvej kapitole. Jeho konštrukcia je založená podľa normy SP800-90 a generátor nesie označenie AES_CTR_DRBG. Zároveň patrí do kategórie CSPRNG. Poskytuje ochranu na úrovni 256-bitov. Na inicializáciu tajnej počiatočnej hodnoty (ďalej ako seed) používa funkciu `df()`.

Ďalšie označenie, ktoré sa používa v systéme je tzv. **zásobníkový PRNG** (z ang. *Buffered PRNG*). Základný RNG nie je používaný priamo, ale prostredníctvom ochrannej vrstvy (z ang. *wrapping layer*). Tá poskytuje generátoru niekoľko vlastností. Konkrétne:

- pridáva malý zásobník – ďalej buffer,

- uzatvára podporu viac-vlaknovosti (z ang. *multi-threading*),
- poskytuje verziu seed-u.

Ukladanie dát do buffer-a sa realizuje priamo a zlepšuje výkon najmä pri generovaní menšieho množstva čísel. Jeho veľkosť je 128 bajtov. Plní sa vždy po použití uvedenej veľkosti dát. Predstavme si, že obsahuje napríklad 10 bajtov. Systém vyšle požiadavku na väčší objem dát ako sa nachádza v zásobníku. V tomto prípade sa použijú spomínané bajty uložené v buffer-i a zvyšok sa dodá po doplnení zásobníka. Tento úkon zabezpečí vyššie opísaný základný generátor. Po použití bajtov dochádza k vynulovaniu príslušných pozícií. Aktualizácia tejto vyrovnávacej pamäte sa uskutočňuje obdobne pri každej aplikácii nového seed-u (ďalej ako **reseed**). Všetky údaje sa prepíšu. V prípade, že požiadavka o náhodné dáta prekračuje veľkosť 128 bajtov, tak generovanie výstupov je uskutočnené priamo základným PRNG. Prístup k stavu buffer-ovaného PRNG je zablokovaný. Tým sa zabezpečuje, že ostatné vlákna nemôžu čítať ani modifikovať rovnaké stavy v rovnakom čase.

Zdrojový PRNG. S takto označeným generátorom sa stretávame pri tvorbe a ukladaní náhodnosti v entropickom systéme, charakterizovanom o podkapitolu nižšie. Je súčasťou CNG.SYS modulu a ide o už vyššie opísaný zásobníkový generátor náhodných čísel.

V režime jadra je použitý opäť zásobníkový PRNG s daným stavom pre logický procesor. To znamená, že ak máme 8 jadrový procesor s podporov tzv. **viacerých vlakien** (z ang. *hyperthreading*), tak máme k dispozícií 16 logických procesorov. V prípade, že aplikácia požiada o náhodne dáta v režime jadra, tak algoritmus skontroluje, ktorý z procesorov je aktívny a overí stav generátora. Ak bol generátor alokovaný na tomto procesore, tak následne sa použije na generovanie náhodných dát. V opačnom prípade dochádza k návratu na pôvodný procesor. Tento CPU sa následne pokúša o alokovanie nového stavu PRNG. Dodanie seed-u zabezpečí zdrojový generátor. Ak proces alokácie nebude úspešný, tak o službu generovania sa postará zdrojový generátor. V praxi však k neúspešnej alokácii nedochádza a náhodné dáta dodá PRNG z CPU. Anglické označenie vyššie opísaného PRNG v OS Windows 10 je tzv. *Kernel per-processor PRNG*.

Pre používateľa sa mení iba spôsob prístupu ku generátoru. Každý proces vytvorený užívateľom, ktorý vytvorí požiadavkou o náhodné dáta, je spracovaný dynamickou knižnicou bcryptprimitives.dll. Tá vytvorí tzv. *ProcessPRNG*. Ním žiada režim jadra o generovanie náhodných bitov, prostredníctvom vyššie opísaného spôsobu. Tento celý proces dokáže zlyhať jedine v prípade, že načítanie

knižnice bcryptprimitives nebude úspešné. V tomto prípade je celý proces zrušený. V praxi sa na bežiacom počítači tento postup opakuje pri každom procese, ktorý vyžaduje náhodné dáta.

Výsledný počet celkovo aktívnych stavov PRNG je možne vyjadriť ako súčin (N+1) a (M+1). N nám označuje množstvo jadier systému a M počet spustených procesov, ktoré vyžadujú náhodné dáta. Takýto súčin nám vytvára celkom veľký výsledok aktívnych PRNG s príslušným stavom. To však už v súčasnosti nie je problém, pretože výkonné počítače obsahujú dostatok pamäte, aby to bez problémov zvládli. V odbornej literatúre sa definuje vyššie uvedený prístup jedného procesu k generátoru ako – z ang. *Process base PRNG*. Pohľad z vyššej perspektívy popisuje výraz – z ang. *Process per-processor PRNG*.

2.4 Entropický systém

Nasledujúci text je parafrázou [28, str. 6]. Dodanie náhodnosti má za úlohu špeciálny nástroj – Entropický systém. V OS Windows 10 pozostáva z niekoľkých častí, resp. komponentov. Tvoria ho konkrétne:

- zdroje náhodnosti – z ang. *Entropy sources*,
- uložiská náhodnosti – z ang. *Entropy pools*, slúžia ako úložisko dát zdrojov entropie, ktoré sa následne používajú ako seed hodnota pre generátory náhodných čísel.
- obnovovacia, resp. resetovacia logiky seed-u (z ang. *Reseed logic*). Jej úlohou je rozhodovať ako a kedy dôjde k aktualizácii seed hodnoty v zdrojovom PRNG z polí náhodnosti.

Proces reseed-u zdrojového generátora sa opakuje periodicky pomocou časového plánovača. Spúšťa sa jednu sekundu po inicializácii OS a následne každým trojnásobkom predchádzajúcej hodnoty (3, 9, 27, ...). Koniec nastane po dosiahnutí časového limitu. Ten je nastavený na jednu hodinu teda 3600 sekúnd. Plánovanie je nastavené tak, aby k reseed-u došlo iba v prípade, ak je CPU prebudený. Dôvodom sú vysoké energetické nároky na prebudenie.

2.4.1 Zdroje náhodnosti

Ich úlohou je poskytovať náhodné dáta. Výstup je následne uložený v uložiskách entropie.

Súčasťou jadra rozhrania systému je rozhranie na vytváranie nových zdrojov entropie. API vo všeobecnosti podporuje dva typy zdrojov:

- nízke – z ang. *low pull*,
- vysoké – z ang. *high pull/push*.

Rozdiely v ich spracovaní sú minimálne. Nízke zdroje označujú nepodmienené udalosti ako je napríklad pohyb myši. Tie však môžu byť reprodukovateľné. Vysokokvalitné náhodné dáta zabezpečujú vysoké zdroje. V prípade potreby dokážu ihneď (okamžite) poskytnúť entropiu všetky z uvedených zdrojov. Rozdiel medzi typom **pull** a **push** je v tom, že prvý spomenutý neobsahuje vlastnú logiku časovania na obnovu entropie. Tieto zdroje sú dodatočne informované pri každom použití zdrojovým RNG.

Prerozdelenie dát v rámci jednotlivých uložísk entropie sa riadi pomocou tzv. z ang. *Round-Robin*[9] plánovania. Pri vysokých zdrojoch sa vždy prvých 32 bajtov ukladá do prvého úložiska entropie a zvyšné sa uložia do poľa, ktoré nasleduje podľa plánu. Viac informácii k zdrojom náhodnosti je dostupných v [28, str. 8].

Použité zdroje [28, str. 8]

Windows 10 používa viacero zdrojov entropie za účelom poskytnúť dobrú entropiu v každej situácii. V odbornej literatúre sa uvádzajú tieto zdroje.

- **Časovače prerušení** – primárny zdroj. Každé prerušenie je spracované pomocou TSC. Je to počítadlo, ktoré beží na procesore. V x86 a x64 architektúrach sa na získanie jeho hodnoty používa inštrukcia RDTSC. Vo vnútri inštrukcie dochádza k rotovaniu a následnému xorovaniu hodnôt predtým než sa poskytujú ďalej.
- **Štart** – pri štarte systému dochádza k zhromažďovaniu TSC údajov pred načítaním CNG modulu. Tento proces typicky sprevádza niekoľko stoviek prerušení. Po inicializácii tohto ovládača sa údaje použijú hneď ako seed pre koreňový generátor. Teda polia entropie sú v toto kroku vynechané. Tento krok zabezpečuje, že systém má po štarte k dispozícii dostatok entropie
- **TPM**[29] - pri štarte dodáva 40 bajtov. Po registrácii zdroja poskytuje 64 bajtov pri každom reseede. Ten sa opakuje raz za 40 minút.

[9] https://en.wikipedia.org/wiki/Round-robin_scheduling

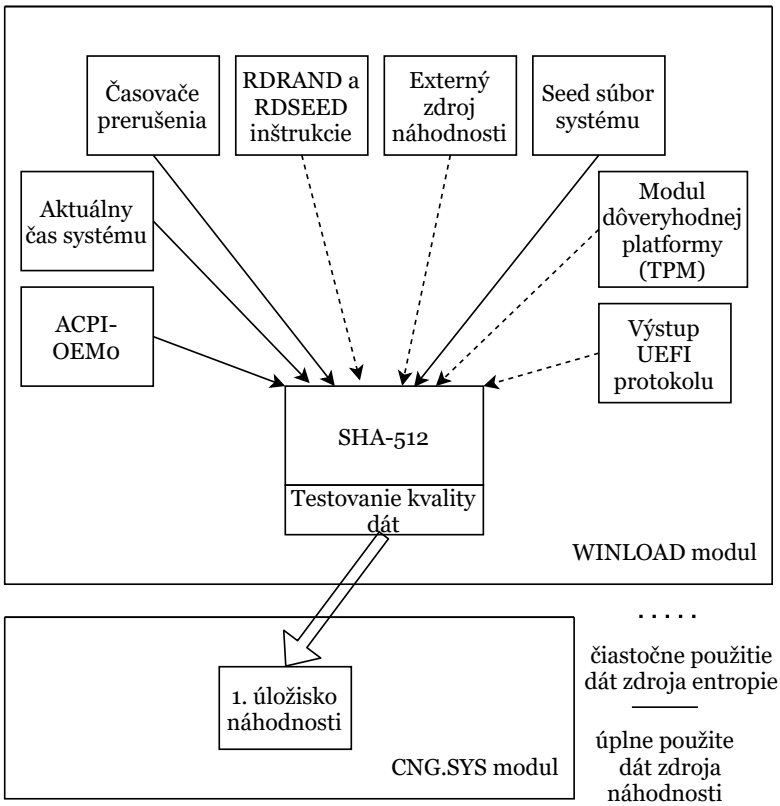
- **RDRAND/RDSEED** - procesorové inštrukcie na tvorbu náhodných dát.
- **Seed súbor** - register v podsystéme registrov. Vytvára ho OS. Používa sa pri ďalšom štarte systému (z ang. *boot*). Nová hodnota je zapísaná pomocou výstupu systémového PRNG. Jeho veľkosť je 64 bajtov. Zapisuje sa pri štarte a vypnutí systému. Ak však dôjde k netradičnému vypnutiu OS, tak zápis prebieha aj počas behu. Štyri minúty po obnovení systému dochádza každým trojnásobkom tohto času k periodickému zápisu až do dosiahnutia limitu ôsmich hodín. Pri vypnutí dochádza k použitiu všetkej uloženej náhodnosti zdrojovým generátorom. Výstup je uložený v tomto súbore a pri nasledujúcom štarte použitý pomocou winload modulu.
- **Externý zdroj** - používateľ má možnosť pridať k zdrojom aj vlastný. Použije sa pri bootovaní systému pomocou winload modulu a po nasadení CNG.SYS ovládača sa tento zdroj po štarte odstraňuje.
- **ACPI-OEM0** - je to ACPI[30] tabuľka s názvom OEM0. Vytvára ju hypervízor Hyper-V. Jej obsahom je 64 bajtov náhodných dát.
- **Firmvérové údaje**,
- **UEFI**[31] protokol,
- **Čas spustenia**.

2.4.2 Úložiská entropie – Entropy pools

Každé pole, respektíve úložisko entropie je implementované pomocou hašovacej funkcie SHA-512, ktorá patrí do rodiny SHA2. To znamená, že všetky dáta tohto poľa sú výstupom SHA funkcie. Ten je pridaný na koniec daného poľa. Vstup hashovacej funkcie tvoria zdroje entropie. Pri štarte systému existuje len jedno takéto úložisko. Po dosiahnutí limitu reseed-u sa zapne možnosť viacerých, resp. dodatočných úložísk entropie. A povolí sa vytvorenie ďalšieho poľa. Dizajn viacnásobného úložiska entropie systému opisuje príloha v dokumente [28]

2.4.3 Štart systému

Pri štarte systému dochádza k prvotnému seedu. Uskutočňuje ho modul Winload pred štartom ovládača Ntoskml. Proces vytvorenia tajnej hodnoty je opísaný v schéme 2.6. Výstup týchto dát je použitý generátorom **AES-CTR-DRBG** podľa normy SP 800-90. 48 bajtov výstupu je použitých modulom CNG.SYS a zvyšok



Obr. 2.6: Schéma vytvorenia prvotného seed-u pri inicializácií systému

je presmerovaný do jadra na budúce použitie. Spomenuté bajty sú použité zdrojovým PRNG. Následne sa spúšťa vytváranie vlastnej entropie pomocou vyššie opísaných zdrojov. Informácie boli čerpané z [28, str. 8]

2.5 Zhrnutie bezpečnosti operačného systému

Aktuálne platnú validáciu FIPS 140-3, OS Windows 10[10] spĺňa aj napriek tomu, že kryptografické rozhranie obsahuje, v niektorých prípadoch aj používa, neschválené, resp. neodporúčané algoritmy. Tie sa v dnešnej dobe naďalej nepovažujú za kryptograficky bezpečné. OS však ponúka riešenie aj pre užívateľov, ktorí vyžadujú použitie aktuálne platných štandardov. Ide o tzv. **Fips režim**. Ten je dostupný v každej edícií Windows-u 10, okrem Home verzie. Postup, ako spustiť daný režim, je dostupný na stránke spoločne so zoznamom validácii nasadených systémov.

[10]Zoznam validácií jednotlivých produktov spoločnosti Microsoft je dostupný na stránke: https://docs.microsoft.com/en-us/windows/security/threat-protection/fips-140-validation

3 Štatistické testovanie generátorov

Pri používaní RNG môže dôjsť k jeho vnútorným zmenám. Na statické overenie kvality generátorov náhodných čísel sa používajú testy náhodnosti. Väčšina z nich preveruje jednu alebo hneď niekoľko štatistických vlastností. Cieľom je odhaliť periódu, respektíve nadväznosť bitov v danej sekvencií dát. Pre používateľa sú k dispozícií vo forme jednotlivých testov alebo štatistických testovacích sád. Takáto zbierka predstavuje viacero implementácií skúšok zoskupených dokopy, pripravených na jednoduché použitie, modifikáciu a vyhodnotenie. Najznámejšími sú:

- **Dieharder**[32],
- **NIST STS**[22] – z ang. *The NIST Statistical Test Suit*.

Okrem vyššie uvedených je však možné vyhľadať, vytvoriť, a aj upraviť, respektíve optimalizovať rôzne implementácie štatistických testov. Dôkazom toho je aj vznik modifikovanej NIST testovacej sady – [33]. Podľa vyjadrení autorov uvedeného projektu sa im podarilo zefektívniť rýchlosť testovania o vyše 50 percent v porovnaní s originálom.

Je však nutné si uvedomiť, že úspešné absolvovanie ľubovoľného počtu štatistických testov nezabezpečuje kryptografickú bezpečnosť daného generátora. Dôvodom je, že niektoré z nástrojov na produkciu náhodných dát môžu byť pripravené tak, aby boli úspešné pri exekúcii týchto testov. Avšak platí, že ak má byť RNG označený za CSPRNG, tak musí úspešne zvládnuť testovanie štatistickými testami.

3.1 Testovacia zbierka Dieharder

Prvá zo spomenutých sád vznikla v roku 2003 a je aj naďalej udržiavaná. Slúži na testovanie konkrétnych RNGs. Teda ak by sme uvažovali o použití tejto testovacej zbierky, tak pre správnosť postupu by bolo nutné aplikovať testovanie na

AES-e v režime, ktorý používa operačný systém Windows na produkciu náhodných dát. Z vyššie uvedených dôvodov charakterizujeme túto sadu veľmi stručne.

Aktuálna verzia – 3.31.1, pozostáva z :

- 17 Diehard,
- 3 NIST STS,
- 10 testov, autora tejto zbierky – *Róberta G. Brown-a*.

Podrobnejší opis týchto testov je dostupný online[1,2]. Inštalácia sady je jednoduchá najmä v prostredí OS Linux. Realizuje ju príkaz: **sudo apt-get install -y dieharder**[3]. V prostredí systému Windows je potrebná zložitejšia konfigurácia.

3.2 NIST – Štatistická testovacia sada

Tento balíček bol vytvorený inštitúciou NIST. Pozostáva z 15 testov. Slúžia na overenie kvality výstupných dát z hardvérových, a aj softvérových generátorov náhodných čísel. V súčasnosti sa práve táto zbierka používa aj pri testovaní CSPRNG. Na rozdiel od spomenutej kolekcie *Dieharder* táto zbierka vyšla s podrobnou dokumentáciou [22]. Uvádzame stručný opis metód a použitých testov.

3.2.1 Obsah sady a opis implementovaných testov

Sada je pre používateľa dostupná vo forme archívu – „**sts-2.1.2.zip**“[4]. Jeho obsahom sú implementácie štatistických testov v jazyku C. **Makefile** na jednoduché vytvorenie spustiteľného programu a príklady vstupných dát z rôznych typov generátorov. Následné spustenie je opísané v [22, kap. 5.6] a pomocou ukážky 3.1.

Opis testov štatistickej sady:

1. **Frekvenčný test – *The Frequency/Monobit Test*[22, kap. 2.1]**

   Výsledok je určený pomerom jednotkových a nulových bitov v danej testovanej sekvencii. Úspešný je vtedy, ak sa obsah jednotiek v našich dátach

[1] https://sites.google.com/site/astudyofentropy/background-information/the-tests/dieharder-test-descriptions

[2] https://en.wikipedia.org/wiki/Diehard_tests

[3] Príkaz pre distribúciu Ubuntu.

[4] Dostupné online: https://csrc.nist.gov/CSRC/media/Projects/Random-Bit-Generation/documents/sts-2_1_2.zip

blíži k $\frac{1}{2}$ z celkového počtu. Odporúčaná dĺžka testovanej postupnosti je minimálne 100 bitov. Úspešné zvládnutie tohto testu je nutnou podmienkou pre ďalšie pokračovanie testovania.

2. **Blokový frekvenčný test – *Frequency Test within a Block*[22, kap. 2.2]**

   Podobný prvému zo spomenutej sady. Namiesto celej testovacej sekvencie dát dochádza k frekvenčnému testu M-bitových blokov. Úspešný je vtedy, keď sa výsledný pomer rovná $\frac{M}{2}$, pričom M je počet bitov jedného bloku. Obdobne aj v tomto prípade sa odporúča testovať minimálne 100 bitovú postupnosť.

3. **Test rovnakých reťazcov – *Run Test*[22, kap. 2.3]**

   Zameraný na celkový počet po sebe idúcich neprerušených a rovnakých bitov – tzv. **behov**. Cieľom tohto testu je overiť rôznorodosť striedajúcich sa sekvencií pri generovaní. Odporúča sa aplikovať na sekvencie s dĺžkou minimálne 100 bitov

4. **Test najdlhšej sekvencie jednotiek v bloku – *Test for the longest Run of Ones in a Block*[22, k. 2.4]**

   Zameraný na najdlhší beh v M-bitových blokoch. Má za úlohu overiť, či vygenerovaná najväčšia sekvencia je v súlade s najdlhšou sériou, aká sa očakáva, resp povoľuje v náhodnej sekvencii bitov[5]. Odporúča sa aplikovať test na minimálne 128 bitov náhodnej postupnosti. [6]

5. **Test série binárnych matic – *Binary Matrix Rank Test*[22, kap. 2.5]**

   Zameraný na poradie disjunktných[7] submatíc celej testovacej sekvencie. Overuje lineárne závislosti medzi podreťazcami s pevnými veľkosťami. Tento test je obdobne súčasťou sady Dieharder. Pre správne fungovanie je minimálna dĺžka sekvencie stanovená na 38 912 bitov.

6. **Test diskrétnej Fourierovej transformácie – *Discrete Fourier Transform* (*Spectral*) *Test*[22, kap. 2.6]**

   Zameraný na výšky vrcholov – **amplitúd**, jednotlivých bitov testovanej postupnosti v diskrétnej Fourierovej transformácii[8]. Cieľom je detekcia periodických znakov, ktorá by naznačovala odchýlku od predpokladu náhod-

[5] Povolený počet jednotiek závisí od veľkosti testovanej sekvencie
[6] V tomto prípade sa dovoľuje maximálne 8, po sebe idúcich, jednotkových bitov.
[7] Disjunktné množiny sú také, ktoré nemajú žiaden spoločný prvok
[8] https://cs.wikipedia.org/wiki/Fourierova_transformace

nosti. Zisťujú sa prahové hodnoty[9] 95% znakov a zvyšných 5% testovanej sekvencie. Následne sa overuje, či nedochádza k významnej odlišnosti medzi týmito hodnotami. 1 000 bitov sa odporúča ako minimálna veľkosť postupnosti.

7. **Test neprekrývajúcich sa vzorov – *Non-overlapping Template Matching Test*[22, kap. 2.7]**

   Zameraný na počet vopred určených bitov v testovanom reťazci – tzv. okne. Cieľom je detegovať generátor, ktorý produkuje veľa neperiodického vzoru. Test používa m-bitové okno na vyhľadanie konkrétneho m-bitového vzoru. Ak ho nenájde, nastane posun okna o jednu bitovú pozíciu. Ak sa nájde, okno sa resetuje na bity po nájdenom vzore a vyhľadávanie pokračuje. Nemá odporúčanú dĺžku vstupnej postupnosti. Parametre pre tento test sa vypočítajú na základe požiadavky.

8. **Test prekrývajúcich sa vzorov – *Overlapping Template Matching Test*[22, kap. 2.8]**

   Pracuje na rovnakom princípe ako 7. test. Rozdiel je len pri zhode vzoru s oknom. V tomto prípade sa okno posúva o jeden bit a následne pokračuje prehľadávanie.

9. **Mauerov „univerzálny štatistický“ test – *Maurer's „Universal Statistical“ Test*[22, kap. 2.9]**

   Zameraný na počet bitov medzi zhodnými vzormi. Overuje, či je možné danú postupnosť komprimovať bez straty informácií. Ak je možná veľká kompresia, tak daná sekvencia sa nepovažuje za náhodnú. Vstupná postupnosť môže mať variabilnú dĺžku. Odporúča sa aplikovať na minimálne 1 000 000-bitovú postupnosť.

10. **Test lineárnej zložitosti – *Linear Complexity Test*[22, k. 2.10]**

    Zameraný na dĺžku posuvného registra s lineárnou spätnou väzbou – LFSR[10]. Cieľom je zistiť, či je daná postupnosť dostatočne zložitá, aby sa mohla považovať za náhodnú. Platí, že dlhšie LFSRs spĺňajú tento predpoklad.

[9] Prahová hodnota – krajná, resp. hraničná hodnota. Získava sa prahovaním. Viac informácií o tejto metóde je dostupných v dokumente [34, k. 1.3]

[10] Výstup tohto registra je lineárne závislý od jeho počiatočného stavu a predchádzajúcich výstupov. Používa sa napríklad v PRNG a prúdových šifrách.

11. **Test Sérií – *Serial Test*[22, kap. 2.11]**

    Zameraný na frekvenciu všetkých možných prekrývajúcich sa m-bitových vzorov v celej testovanej postupnosti. Cieľom je zistiť, či počet výskytov je približne rovnaký, ako by mal byť v náhodnej sekvencii. Tá by mala byť rovnomerná[11]. Vyžaduje sa zvoliť m také, pre ktoré platí $m < [log_2 n] - 2$. Veľkosť vstupnej sekvenciu v bitoch reprezentuje premenná n.

12. **Približný test Entropie – *Approximate Entropy Test*[22, kap. 2.12]**

    Podobný ako Test Sérií. Zameraný na frekvenciu všetkých možných prekrývajúcich sa vzorov s dĺžkou m-bitov. Porovnanie sa aplikuje na frekvencie dvoch po sebe nasledujúcich blokov, ktoré sa prekrývajú. Výsledok sa následne porovná s ekvivalentom pre náhodnú sekvenciu. Vyžaduje sa zvoliť m také, pre ktoré platí $m < [log_2 n] - 5$. Veľkosť vstupnej sekvenciu v bitoch reprezentuje premenná n.

13. **Test kumulatívnych súčtov – *Cumulative Sums Test*[22, kap. 2.13]**

    Zameraný na maximálnu odchýlku od nuly pri kumulatívnom súčte všetkých bitov. Pri tomto postupnom hromadnom sčítaní predstavujú jednotkové bity kladné jednotky. Nuly na druhej strane záporné. Cieľom testu je určiť, či takýto kumulatívny súčet testovaných dát zodpovedá náhodnej sekvencii. Test je úspešný, ak sa výsledok sčítania blíži k nule. Odporúča sa, aby každá testovaná sekvencia mala minimálnu dĺžku 100 bitov.

14. **Test náhodných návštev – *Random Excursions Test*[22, kap. 2.14]**

    Zameraný na počet cyklov, ktoré majú presne K náhodných návštev v kumulatívnom súčte. Všetky cykly majú dĺžku zvolenú náhodne. Cieľom je zistiť či sa počet návštev odlišuje od hodnoty platnej pre náhodné dáta. Test pozostáva z 8 čiastočných testov. Každý otestuje jeden zo stavov: - 4, - 3, - 2, - 1, 1, 2, 3, 4. Úspešný je iba ak dáta uspejú vo všetkých čiastočných testoch. Odporúča sa testovať postupnosť s minimálnou dĺžkou 1 000 000 bitov.

15. **Test variantov náhodných návštev – *Random Excursions Variant Test*[22, kap. 2.15]**

    Zameraný na celkový počet návštev jednotlivých stavov pri kumulatívnom súčte. Pozostáva z 18 testov. Postupne sa testujú stavy: - 9, - 8, ..., - 1, 1, ..., 8, 9. Odporúčaná minimálna veľkosť vstupu je rovnaká ako v 14. teste.

[11] Rovnomernosť, resp. uniformita, náhodnej sekvencie znamená, že každý m-bitový vzor ma rovnakú pravdepodobnosť objaviť sa, ako ktorýkoľvek iný.

3.2.2 Vyhodnotenie výsledkov štatistických testov

Nasledujúce vety vznikli parafrázou [22, kap. 1.1.5]. Testovanie pomocou sady je založené na overení dvoch predpokladov, tzv. **nulovej** a **alternatívnej** hypotézy. Prvá z uvedených, ozn. H_0, tvrdí, že testovaná postupnosť je náhodná. Druhú definujeme ako inverznú voči nulovej, teda výstup z RNG je nenáhodný. Len jeden z týchto predpokladov je prijatý v priebehu aplikácie sady. V súvislosti s týmto postupom môže dôjsť k 2 typom chýb v priebehu testovania.

1. Sekvencia je náhodná, ale H_0 nebola akceptovaná.

2. Postupnosť nie je náhodná, avšak nulová hypotéza je prijatá.

Pravdepodobnosť, že dôjde k prvému zo spomenutých dejov definuje pojem – **hladina významnosti**, označíme α. Jej veľkosť závisí od konkrétneho štatistického testu. Typicky sa volí z intervalu $[0.001; 0.01]$

Na vyhodnotenie uvedených predpokladov dochádza pri každom teste k porovnaniu **výslednej štatistickej** – S,[12] a **kritickej** hodnoty, ďalej CV[13]. Každý z testov má definované štatistické hodnoty, na základe ktorých môžeme prijať alebo odmietnuť H_0. CV je hodnota získaná štatistickým rozdelením hodnôt. Platí, že predstavuje výsledok testu, ktorý pochádza na 99% z nenáhodnej sekvencie testovaných dát. Inými slovami, predstavuje hodnotu, pri ktorej už neakceptujeme H_0 ako pravdivé. Program vytvorí výpis s medzi-výpočtami pre každý test a zároveň aj finálny výsledok testu. Používateľ si však môže všimnúť tzv. **pravdepodobnostné hodnoty** – $p - values$. Získa ich výpočtom pravdepodobnosti z vyššie uvedeného porovnania. Ten reprezentuje pravdepodobnosť, vygenerovania lepšej náhodnej postupnosti dát ako by zvládol dokonalý RNG. Pri úspešnom teste platí (3.1).
Nech

$$X = p - values = P(S),$$

potom,

$$(X \in (\alpha < X \geq 1)) \Leftrightarrow (H_0 = pravda). \tag{3.1}$$

3.2.3 Doba vykonania testovacej sady

Na meranie dĺžky sme modifikovali zdrojové kódy uvedenej sady. Použité bolo Windows rozhranie na meranie času – **QueryPerformanceCounter()**. Taktiež sme zistili počet cyklov jednotlivých testov vzhľadom na veľkosti vstupov. K

[12] Hodnota získaná aplikovaním daného testu na naše dáta
[13] Z ang. Critical Value

tomu nám dopomohli inštrukcie **RDTSC** a **RDTSCP**. Opis použitých metód je obsahom kapitoly 4.

Experimentálne výsledky sú znázornené v tabuľke 3.1. Výsledná doba vykonania daného testu je závislá na veľkosti sekvencie. Pre celkový čas platí (3.2). Ak by sme si vyhradili procesor iba pre nás, tak potom jednotlivé časy t_i by boli konštantné. Dôvodom je deterministický charakter procesu. Tento úkon však nie je v používateľskom režime možný. Uvedený vzťah ráta s týmto faktom.

Nech

n – veľkosť vstupnej testovacej postupnosti,

t_i – doba aplikovania všetkých testov na n-bitovú sekvenciu,

m – počet testovaných prúdov s veľkosťou n,

T – výsledný čas celého testovania.

Potom

$$T = \sum_{i=1}^{m} t_i. \tag{3.2}$$

Test č.	**Veľkosť vstupnej testovanej sekvencie**			
	122kB ≈ 1 000 000 b		**1,165GB ≈ 10 000 000 000 b**	
	ČAS [s]	**CYKLY**	**ČAS [s]**	**CYKLY**
1	0,004 055	11 730 471	5,387 021	15 593 089 692
2	0,002 740	7 921 031	3,591 694	10 396 392 294
3	0,009 644	27 907 135	13,367 284	38 692 505 860
4	0,014 938	43 230 329	21,562 572	62 414 330 729
5	0,007 866	22 760 418	10,895 430	31 537 547 299
6	0,074 364	215 241 654	104,454 277	302 350 042 366
7	0,226 567	655 801 389	0,036 371	105 257 157
8	1,763 558	5 104 725 551	2 485,426 270	719 423 6071 645
9	0,056 390	163 214 697	78,888 596	228 348 426 960
10	0,045 201	130 830 194	67,123 001	194 292 108 778
11	0,203 056	587 748 800	285,328 339	825 902 376 471
12	0,004 447	12 862 109	10,411 473	30 136 705 606
13	0,005 211	15 068 574	76,817 490	222 353 456 121
14	0,564 533	1 634 064 740	781,071 777	2 260 865 638 374
15	3,703 921	10 721 233 139	5 222,643 066	15 117 297 898 720
Sumár	6,686501	19 354 524 033	9 167,004 883	26 534 522 105 621

Tabuľka 3.1: Meranie testovania sady pomocou konfigurácie A

Odporúčanie pri spustení

Na spustenie sady je potrebná inicializácia. Používateľ ju realizuje vstupmi z príkazového riadku. Zdrojový kód 3.1, znázorňuje tento úkon. Veľkosť otestovaných dát pomocou takejto konfigurácie je 11,92 MB[14].

Pre optimálne fungovanie všetkých testov je ideálnejšie použiť veľkosť vstupnej sekvencie rovnú 1 000 000 bitov teda približne 122kB. Následne pri väčšom objeme testovaných dát zvolíme vyšší počet prúdov. Týmto postupom zamedzíme vzniku chýb pri alokovaní potrebnej pamäte a vykonané testy poskytnú kvalitnejšie výsledky.

Zdrojový kód 3.1: Ukážka spustenia sady NIST STS

```
1 > ./assess.exe 1000000   // size of random sequence in bits
2 > 0                      // apply tests on input file
3 > myRandomDataFile.bin   // name of file with random numbers
4 > 1                      // apply all tests
5 > 0                      // no changes of default tests values
6 > 100                    // number of bit streams
7 > 1                      // choosen bin format of input file
```

[14] 1 000 000 b $*$ 100 $=$ 100 000 000 b $\approx$ 11,92MB

4 Metodika testovania dát a merania rozhraní

Pred samotným generovaním dát bolo nutné zvoliť si meracie a testovacie metódy. V tejto kapitole si ich popíšeme.

Implementáciu rozhraní sme realizovali v programovacom jazyku C. Preklad do strojového kódu zabezpečí prekladač GCC vo verzii 10.2.0. **Overenie kvality dát** vykonáme pomocou NIST STS, opísanej v podkapitole 3.2. Pri experimentálnych meraniach implementácií RNG rozhraní sme sa zamerali na tri údaje:

- čas exekúcie samotnej implementácie – T_A,
- dobu vykonania, vrátane bežných úkonov[1] – T_B
- priemerný počet cyklov API – ANC[2]

Pomocou týchto nameraných hodnôt sme vypočítali priepustnosť dát. Použili sme vzťah (4.1).
Nech

$_X = \{A, B\}$,

T_X – čas zvoleného procesu,

NI – počet opakovaných volaní[3],

BS – veľkosť zásobníka[4],

$V_D = NI * BS$ – celkový objem vygenerovaných dát,

P_X – priepustnosť procesu,

potom,

$$P_X = \frac{V_D}{T_X} \tag{4.1}$$

[1] Ukladanie dát, overenie úspešnosti generovania, ...

[2] Predstavuje priemernú hodnotu. Získa sa ako pomer súčtu všetkých vykonaných cyklov a počtu opakovaní volania testovanej funkcie. Vypočítané iba v prípade T_A

[3] Z ang. *Number of iterations*

[4] Z ang. *Buffer Size*

Uvedené experimenty boli aplikované na troch zariadeniach. Špecifikáciu týchto prenosných počítačov znázorňuje tabuľka 4.1. Všetky notebook-y sme počas testovania pripojili do elektrickej siete. Režim napájania sme zmenili na „Vysoký výkon“.

Označenia hodnôt a meracích nástrojov, ktoré vznikli v tejto kapitole **sú použité pri interpretácii výsledkov**.

Komponenty	Konfigurácia		
	A – ASUS TUF A15	**B – ASUS Vivo15**	**C – LENOVO IdeaPad**
Model	F506IU-AL006T	X510UN-BQ148R	S540-15IML
Verzia OS	Win 10 Home; 64-bit.; v.20H2	Win 10 Pro; 64-bit.; v.2004	Win 10 Home; 64-bit.; v. 20H2
Zostava OS	19042.964	19041.928	19042.985
CPU	AMD Ryzen 7 Mobile 4800H	Intel Core i5-8250U	Intel Core i5-10210U
RAM	16 GB DDR4 2x1600 MHz	8 GB DDR4 1x2400 MHz	8 GB DDR4 2x1200 MHz
Úložisko	SSD OM8PCP3512F-AB	HDD MQ04ABF100	SSD SSDPEKNW512G8L

Tabuľka 4.1: Technická špecifikácia použitých počítačov

V tomto bode treba spomenúť určitú odchýlku experimentálnych meraní od skutočných hodnôt. Najväčším faktor, ktorý mohol spôsobiť chybu merania sú prerušenia operačného systému. [5]. Uvedenému procesu je možné sa vyhnúť jedine implementáciou modulu pre prácu v režime jadra. Následne by sme si vyhradili procesor pre vlastné účely. Kladom módu by mohol byť aj menší počet inštrukcií potrebných na vykonanie funkcionality. S týmto postupom sa však používateľ často nestretáva. Uvedený fakt je dôvodom, prečo sme sa pri meraniach zamerali iba na beh rozhraní v používateľskom režime. Pripomeniem však, že rozhrania v oboch režimoch poskytujú **identické služby**.

4.1 Časové meranie rozhraní

Pri meraní dĺžky behu rozhrania sme implementovali Windows API:

- `QueryPerformanceCounter()` –QPC [35],
- `QueryPerformanceFrequency()` –QPF [36].

Informácie o funkciách sú dostupné vo forme webovej dokumentácie [37]. Použitie týchto rozhraní pri meraní znázorňuje zdrojový kód 4.1. Uvedeným postupom dokážeme odmerať čas vykonania algoritmu s presnosťou nanosekúnd[6]. Presnosť

[5] Viď. napríklad elapsedTime v 4.1

[6] API je možné implementovať s presnosťou piko-sekúnd. Viď [37, Using QPC in native code]

metódy sa dá overiť jednoducho. Napríklad pomocou funkcie `Sleep()`[7]. Stačí ju vložiť do priestoru merania.

Zdrojový kód 4.1: Ukážka použitia QPC/QPF

```
1  #include<windows.h>
2  #define TIMER_INIT \
3  LARGE_INTEGER frequency; \
4  LARGE_INTEGER t1,t2; \
5  double elapsedTime; \
6  QueryPerformanceFrequency(&frequency);
7
8  // Use to start the performance timer
9  #define TIMER_START QueryPerformanceCounter(&t1);
10 // Use to stop the timer
11 #define TIMER_STOP \
12 QueryPerformanceCounter(&t2); \
13 elapsedTime=(double)(t2.QuadPart-t1.QuadPart)/frequency.QuadPart;
14 int main(){
15   TIMER_INIT
16   {TIMER_START
17     functionMeasurment(); // code for measurment
18     TIMER_STOP}
19   printf("Time␣of␣execution:␣%f␣sec\n", elapsedTime);
20 }
```

Odporúčanie pri pretypovaní dát

V jazyku C môže pri pretypovaní dát dochádzať k zmene poradia vygenerovaných reťazcov. Dôvodom je zmena endianity[8], respektíve uloženia dát v pamäti. S týmto problémom sme sa stretli pri zmene z **unsigned int** na **unsigned char**. Dáta v pamäti boli uložené metodikou malý endián, ale správne poradie vygenerovaných dát reprezentoval veľký endián. V tomto prípade bolo nutné vykonať konverziu endianity. Použili sme na to funkcie znázornené pomocou zdrojového kódu 4.2.

[7] Sleep(1000) reprezentuje jednu sekundu

[8] `https://sk.wikipedia.org/wiki/Endianita`

Zdrojový kód 4.2: Ukážka pretypovania premenných

```
#include<stdio.h>
#define IS_BIG_ENDIAN (!*(unsigned char *)&(uint16_t){1})
int main (){
  unsigned int a = 2343352;
  unsigned char b;
  if(!IS_BIG_ENDIAN)
  castUintToByte(a,b);
  else b=(unsigned char)a;
  printf("%lu",b);
  return 0;
}
```

4.2 Meranie počtu cyklov rozhraní

Obdobne sme sa zamerali aj na počet cyklov rozhraní. Tento údaj nám z pohľadu dlhodobého vývoja predstavuje kvalitnejšiu informáciu ako čas vykonania. Na základe týchto údajov vieme určiť napríklad či v priebehu času došlo k zefektívneniu algoritmov rozhrania[9]. Ďalším príkladom je určenie pomeru prerušení OS a mnoho iných. Meranie sme prvotne realizovali pomocou funkcie ***cpucycles***()[10]. Zdrojový kód 4.3, ju definuje. Táto metóda však neposkytovala dostatočne kvalitné výsledky.

Zdrojový kód 4.3: Meranie počtu cyklov pomocou funkcie cpucycles()

```
int64_t cpucycles(void)
{
  uint64_t result;
  __asm__ volatile(".byte␣15;.byte␣49;shlq␣$32,
␣␣%%rdx;orq␣%%rdx,%%rax"
  : "=a" (result) ::  "%rdx");
  return result;
}
```

Meranie počtu vykonaných cyklov sme, kvôli tomuto faktoru, uskutočnili podľa metód v Intel dokumente [38, kap.3.2.1]. Ukážkou aplikovaného riešenia je kód

[9] Zmenší sa počet cyklov, potrebných na vykonanie.

[10] Funkcia prebraná z git archívu: `https://github.com/newhopecrypto/newhope/tree/master/ref`

4.4.

Zdrojový kód 4.4: Meranie počtu cyklov Intel metódou

```
#include <stdio.h>

//cpucyclesS -- Start measure
static __inline__ uint64_t cpucyclesS(){
  unsigned cycles_low, cycles_high;
  __asm__ volatile ("CPUID\n\t"
  "RDTSC\n\t"
  "mov␣%%edx,␣%0\n\t"
  "mov␣%%eax,␣%1\n\t": "=r" (cycles_high), "=r" (cycles_low)::
  "%rax", "%rbx", "%rcx", "%rdx");
  return  (((uint64_t)cycles_high << 32) | cycles_low );
}
//cpucyclesE -- End measure
static __inline__ uint64_t  cpucyclesE(){
  unsigned cycles_low, cycles_high;
  __asm__ volatile ("RDTSCP\n\t"
  "mov␣%%edx,␣%0\n\t"
  "mov␣%%eax,␣%1\n\t"
  "CPUID\n\t": "=r" (cycles_high), "=r" (cycles_low)::
  "%rax", "%rbx", "%rcx", "%rdx");
  return (((uint64_t)cycles_high << 32) | cycles_low );
}
int main(){
  uint64_t tick,tock;
  tick=cpucyclesS();
  codeForMeasurment();
  tock= cpucyclesE() - tick;
  printf("Executed:␣%llu␣cycles\n", tock);
  return 0;
}
```

5 Generovanie náhodných dát

Na produkciu náhodných dát má používateľ k dispozícii hneď niekoľko možností. V rámci tejto práce si rozhrania rozdelíme do dvoch kategórií.

- Hardvérové API – závisí od technického vybavenia počítača.
- Softvérové API – implementáciu služby RNG zabezpečuje OS alebo knižnice zvoleného jazyka.

V tejto kapitole uvedené rozdelenie viac charakterizujeme pomocou nasledujúcich podkapitol.

5.1 Hardvérové rozhrania

Súčasťou moderných procesorov sú aj implementácie generátorov skutočne náhodných čísel. Ich použitie je realizované pomocou procesorových inštrukcií ***RDRAND a RDSEED***. Práca s inštrukciami si vyžaduje znalosti nízko-úrovňových programovacích jazykov, akým je napríklad *Asembler*[1]. Tie nie sú v dnešnej dobe veľmi populárne. Aj dôsledkom toho výrobcovia procesorov sprístupňujú programátorom tzv. **programovateľné rozhrania – API**. Štandardne sú napísané pomocou vysoko-úrovňových jazykov[2]. Ich transformáciu do strojového kódu zabezpečuje prekladač.

V tejto súvislosti spomenieme dvoch výrobcov procesorových čipov s najväčším zastúpením na trhu – **AMD a Intel**. Obe firmy sprístupnili používateľom svoje API v programovacom jazyku C. Najpodstatnejším rozdielom pri používaní je vzájomná kompatibilita. Intel rozhranie poskytuje podporu iba pre vlastné procesory, zatiaľ čo spoločnosť AMD uverejnila sadu, ktorá má podporu aj iných výrobcov procesorových čipov. Podmienkou úspešného generovania však ostáva potrebná implementácia inštrukcií RDRAND a RDSEED. Z tohto dôvodu sme

[1] `https://en.wikipedia.org/wiki/Assembly_language`
[2] Jazyk C, Java, C Sharp a iné

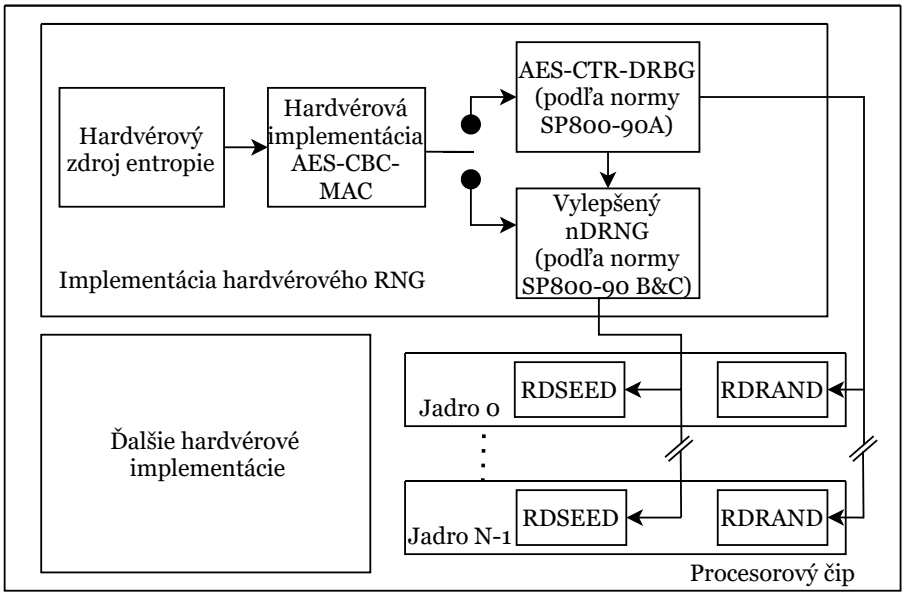
pri generovaní náhodných dát, pomocou vyššie spomenutých príkazov, použili práve riešenie firmy AMD.

V súvislosti s inštrukčnými sadami, implementovanými na dnešných mikroprocesoroch, dávame čitateľovi do popredia dokument [39]. Jeho obsahom sú všetky sady používané na rôznych typoch procesorov. Dokument je pravidelne aktualizovaný a udržiavaný.

5.1.1 RDRAND a RDSEED

Inštrukcie vytvorila spoločnosť Intel. Ich vznik sa datuje do roku **2012**. Samotní autori ([40]), odôvodnili čipovú implementáciu TRNG ako nutný bezpečnostný prvok, ktorý dokáže v akejkoľvek situácii dodať skutočne náhodné dáta. Samozrejme nezávislé od deterministických procesov bežiaceho softvéru.

Inštrukcia RDSEED je implementáciou n-DRNG. Na druhej strane RDRAND realizuje CSPRNG, ktorého zdrojom entropie je výstup RDSEED inštrukcie. Obidva uvedené typy RNG boli opísané v kapitole 1 tejto práce. Proces generovania náhodných dát pomocou týchto príkazov je znázornený na obrázku 5.1[3]. Pod-



Obr. 5.1: Implementácia procesorových inštrukcií RDRAND a RDSEED

robný opis aplikovaných metód v generátore je dostupný v dokumentácii [41].

Podporu týchto inštrukcií doplnila spoločnosť AMD v roku 2015. Základom bola implementácia Intelu. Viac podrobností o riešení sa nachádza v online dokumente [42].

[3]Prebraté z: [40, kap. 3.1]

AMD Secure RNG API [42, str. 1-7]

Voľné dostupné rozhranie[4] v programovacom jazyku C. Aktuálna verzia – 3.0.6, obsahuje celkovo 14 funkcií. Z toho dve na kontrolu implementácie inštrukcií a 12 na produkciu náhodných dát. Programátor má možnosť vygenerovať jedno 16/32/64-bitové číslo alebo ich pole. Obdobne je k dispozícii možnosť zvoliť si požadovanú veľkosť výstupu v bajtoch. Pri našom testovaní sme implementovali práve túto možnosť. Zdrojový kód, 5.1, uvádza deklarácie spomenutých implementovaných funkcií. Pomenovania sú jednoznačné. Argument N označuje počet opakovaní a *retry_count* udáva počet pokusov v prípade zlyhania.

Obsahom balíka je aj zdrojový kód **secrng_test.c**. Ten realizuje príklad použitia každého AMD rozhrania.

[4] `https://developer.amd.com/amd-aocl/rng-library/`

Zdrojový kód 5.1: Funkcie v AMD Secure RNG rozhraní

```
int is_RDRAND_supported();
int get_rdrand16u(uint16_t* rng_val, unsigned int retry_count);
int get_rdrand32u(uint32_t* rng_val, unsigned int retry_count);
int get_rdrand64u(uint64_t* rng_val, unsigned int retry_count);
int get_rdrand32u_arr(uint32_t* rng_val,
                      unsigned int N,
                      unsigned int retry_count);
int get_rdrand64u_arr(uint64_t* rng_arr,
                      unsigned int N,
                      unsigned int retry_count);
int get_rdrand_bytes_arr(unsigned char *rng_arr,
                         unsigned int N,
                         unsigned int retry_count);
int is_RDRAND_supported();
int get_rdseed16u(uint16_t* rng_val, unsigned int retry_count);
int get_rdseed32u(uint32_t* rng_val, unsigned int retry_count);
int get_rdseed64u(uint64_t* rng_val, unsigned int retry_count);
int get_rdseed32u_arr(uint32_t* rng_arr,
                      unsigned int N,
                      unsigned int retry_count);
int get_rdseed64u_arr(uint64_t* rng_arr,
                      unsigned int N,
                      unsigned int retry_count);
int get_rdseed_bytes_arr(unsigned char *rng_arr,
                         unsigned int N,
                         unsigned int retry_count);
```

Výsledky experimentálnych meraní

Dosiahnuté namerané výsledky sú reprezentované formou tabuliek:

- 5.1 – RDRAND a
- 5.2 – RDSEED.

Počítač	Špecifikácie meraní							
	NI	BS	V_D	ANC	T_A [s]	T_B[s]	$\approx P_A$ [MB/s]	$\approx P_B$ [MB/s]
A	1 024	16 kB	16 MB	5 628 973	1,991 472	1,953 684	8,034	8,190
	1 024	16 MB	16 GB	5 676 879 262	2 008,291 504	2 015,468 384	8,158	8,129
	1 024	32 MB	32 GB	11 354 590 480	4 016,876 709	4 050,146 973	8,158	8,091
	8	ULONG_MAX	32 GB	1 453 162 191 677	4 016,253 662	4 051,736 572	8,159	8,087
B	1 024	16 kB	16 MB	6 241 786	3,551 908	3,639 144	4,505	4,397
	1 024	16 MB	16 GB	6 440 954 785	3 664,190 186	3 685,320 801	4,471	4,445
	1 024	32 MB	32 GB	12 897 904 573	7 337,478 516	7 486,955 566	4,466	4,377
	8	ULONG_MAX	32 GB	1 650 387 205 662	7 335,057 129	7 598,942 383	4,467	4,312
C	1 024	16 kB	16 MB	4 805 709	2,330 819	2,351 972	6,865	8,803
	1 024	16 MB	16 GB	4 810 432 963	2 332,328 369	2 338,702 881	7,025	7,006
	1 024	32 MB	32 GB	9 617 320 340	4 662,936 523	4 707,775 879	7,029	6,960
	8	ULONG_MAX	32 GB	1 232 428 756 993	4 668,283 203	4 790,067 383	7,019	6,841

Tabuľka 5.1: Výsledky meraní funkcie get_rand_bytes_arr

Počítač	Špecifikácie meraní							
	NI	BS	V_D	ANC	T_A [s]	T_B[s]	$\approx P_A$ [MB/s]	$\approx P_B$ [MB/s]
A	1 024	16 kB	16 MB	24 876 680	8,800 674	8,673 107	1,818	1,845
	1 024	16 MB	16 GB	25 529 272 494	9 031,408 203	8 902,758 789	1,814	1,840
	1 024	32 MB	32 GB	50 962 566 174	17 528,861 328	17 621,734 375	1,869	1,860
	8	ULONG_MAX	32 GB	6 374 627 781 381	17 618,212 891	18 180,208 984	1,860	1,802
B	1 024	16 kB	16 MB	6 390 445	3,636 515	4,014 131	4,400	3,990
	1 024	16 MB	16 GB	6 447 545 248	3 667,939 209	3 675,636 963	4,467	4,457
	1024	32 MB	32 GB	13 112 239 637	7 459,411 621	7 479,892 578	4,393	4,381
	8	ULONG_MAX	32 GB	1 688 133 062 298	7 502,824 219	7 615,551 758	4,367	4,303
C	1 024	16 kB	16 MB	4 811 400	2,333 592	2,367 678	6,856	6,758
	1 024	16 MB	16 GB	4 819 373 956	2 336,663 330	2 338,929 688	7,012	7,005
	1 024	32 MB	32 GB	9 615 872 296	4 662,234 375	4 706,395 508	7,028	6,962
	8	ULONG_MAX	32 GB	1 231 308 014 422	4 664,037 598	4 769,422 363	7,026	6,870

Tabuľka 5.2: Výsledky meraní funkcie get_rdseed_bytes_arr

5.2 Rozhrania operačného systému Windows

Používateľ má v aktuálnom OS Windows[5] prístup k trojici funkcií realizujúcich službu RNG[28, str. 5].

- **RtlGenRandom**[43]
- **CryptGenRandom**[44]
- **BCryptGenRandom**[45]

Vyššie uvedené realizujú generovanie náhodných čísel pomocou používateľského rozhrania – **ProcessPrng**. V prípade kernel režimu je produkcia dát vykonaná primárne pomocou **SystemPrng** API. Tie boli spomenuté v 2.2. Obsahom tejto podkapitoly je opis týchto funkcií. Ten bol vytvorený na základe webovej dokumentácie spoločnosti Microsoft.

[5]Windows 10 Home - 64-bit, verzia 20H2, zostava OS – 19042.964

5.2.1 RtlGenRandom

Funkcia je deklarovaná v hlavičkovom súbore **ntsecapi.h**, ale nemá knižnicu, ktorá ju vykonáva. Slúži na generovanie pseudonáhodných čísel. Pomocou makra je definovaná ako `SystemFunction036` a až táto je realizovaná v dynamickej knižnici **Advapi32.dll**. Pri použití je teda potrebné načítať tento modul. V súčasnosti je tento krok automatizovaný. Generovanie je realizované použitím rozhrania **ProcessPrng**. Microsoft však odporúča namiesto používania tejto funkcie použitie `CryptGenRandom`.

Špecifikácia [43]

`RtlGenRandom()` potrebuje dva vstupné parametre.

1. `PVOID RandomBuffer` – adresa premennej na uloženie náhodnosti.
2. `ULONG RandomBufferLength` – veľkosť prvého parametra.

Funkcia je typu `boolean`. Teda návratové hodnoty sú `TRUE/FALSE` pri úspechu, resp. neúspechu generovania. Príkladom použitia je zdrojový kód 5.2.

Zdrojový kód 5.2: Príklad použitia RtllGenRandom

```
1  #include<stdio.h>
2  #include<windows.h>   // for variable types definition
3  #include<ntsecapi.h>    // declaration RtlGenRandom()
4
5  int main(){
6    BYTE *pbData=(BYTE*)malloc(sizeof(BYTE) * 10);
7    if(RtlGenRandom(pbData,10) == TRUE){
8      for (int i = 0; i < 10; i++){
9        printf("%u",pbData[i]);
10     }
11     free(pbData);
12     return 0;
13   }
14   free(pbData);
15   return -1;
16 }
```

Výsledky experimentálnych meraní

Dosiahnuté výsledky meraní sú znázornené pomocou tabuľky 5.3.

Počítač	Špecifikácie meraní							
	NI	BS	V_D	ANC	T_A [s]	T_B[s]	$\approx P_A$ [MB/s]	$\approx P_B$ [MB/s]
A	1 024	16 kB	16 MB	15 074	0,005 451	0,020 702	2 935,241	772,872
	1 024	16 MB	16 GB	16 907 537	5,982 000	166,227 746	2 738,883	98,564
	1 024	32 MB	32 GB	33 698 517	11,922 174	524,159 006	2 758,492	62,515
	8	ULONG_MAX	32 GB	4 489 111 941	12,407 025	500,792 203	2 641,084	65,432
B	1 024	16 kB	16 MB	10 580	0.006 771	0.021 564	2 363,019	741,977
	1 024	16 MB	16 GB	14 072 032	8,006 607	136,472 897	2 046,310	120,053
	1 024	32 MB	32 GB	29 400 765	16,726 991	984,013 108	1 958,990	33,300
	8	ULONG_MAX	32 GB	3 940 128 574	17,511 696	275,192 794	1 871,207	119,073
C	1 024	16 kB	16 MB	14 630	0,007 994	0,024 424	2 001,501	655,093
	1 024	16 MB	16 GB	11 823 989	5,733 827	217,114 585	2 857,428	75,462
	1 024	32 MB	32 GB	23 913 126	11,595 265	513,567 815	2 825,981	63,805
	8	ULONG_MAX	32 GB	3 347 779 853	12,680 974	561,609 795	2 584,029	58,347

Tabuľka 5.3: Výsledky meraní funkcie RtlGenRandom

5.2.2 CryptGenRandom

Funkcia na generovanie kryptograficky bezpečných náhodných dát. Vznikla pri prvom riešení kryptografického rozhrania – CAPI. **Zastaraná**, ale zatiaľ podporovaná aj v súčasnom CNG. Microsoft však **neodporúča** jej používanie z dôvodu možného odstránenia v budúcnosti. Jej deklarácia je obsahom hlavičkového súboru **wincrypt.h**. Realizuje ju dynamická knižnica **Advapi32.dll**. Následne je použitý modul **brcyptprimitives.dll**.

Špecifikácia [44]

Pri inicializácii je potrebná trojica parametrov:

1. `HCRYPTPROV hProv`[6] – popis CSP[7], vytvorený funkciou `CryptAcquireContext`,
2. `DWORD dwLen` – veľkosť výstupu – maximum `ULONG_MAX`,
3. `BYTE *pbBuffer` – adresa úložiska, veľkosť musí byť najmenej `dwLen`.

CryptGenRandom je typu `BOOL`. Návratová hodnota je `TRUE`, resp. `FALSE`. V prípade zlyhania je dôvod zapísaný do CSP premennej.

[6] Nutná inicializácia tejto premennej
[7] Popísané v 2.1.1

Zdrojový kód 5.3: Ukážka použitia funkcie CryptGenRandom

```
#include<stdio.h>
#include<windows.h>

int main(){
  HCRYPTPROV   hCryptProv;
  BYTE   *pbData=(BYTE*)malloc(sizeof(BYTE)* 10);
  CryptAcquireContext(&hCryptProv,NULL,
    "Microsoft␣Base␣Cryptographic␣Provider␣v1.0",
                      PROV_RSA_FULL,
                      CRYPT_VERIFYCONTEXT);
  if(CryptGenRandom(hCryptProv,10,pbData)!=0)
  {
    printf("Random␣sequence␣generated.␣\n");
  }
  else
  {
    printf("Error␣during␣CryptGenRandom.\n");
    free(pbData);
    return -1;
  }
  free(pbData);
  return 0;
}
```

Výsledky experimentálnych meraní

Výsledky experimentov funkcie CryptGenRandom znázorňuje tabuľka 5.4.

5.2.3 BCryptGenRandom

Implementácia služby generovania náhodných čísel v druhej verzii kryptografického rozhrania. Ako jediná z uvedených je navrhnutá na použitie v používateľskom aj kernel režime. **Deklarácia** je uvedená v hlavičkovom súbore **bcrypt.h**. Funkcia je následne vykonaná pomocou **bcrypt.dll**. Knižnicu je potrebné pri kompilácií programu prilinkovať. Tento úkon realizujeme pomocou prepínača `-lbcrypt`.

Počítač	Špecifikácie meraní							
	NI	BS	V_D	ANC	T_A [s]	T_B[s]	$\approx P_A$ [MB/s]	$\approx P_B$ [MB/s]
A	1 024	16 kB	16 MB	14 242	0,006 430	0,022 011	2 488,336	726,909
	1 024	16 MB	16 GB	16 991 496	6,013 020	143,722 924	2 724,754	113,997
	1 024	32 MB	32 GB	33795736	11,957515	480,253462	2 740,369	68,231
	8	ULONG_MAX	32 GB	4 525 421 725	12,508601	452,834846	2 619,637	72,362
B	1 024	16 kB	16 MB	10 545	0,008 252	0,021 986	1 938,924	727,736
	1 024	16 MB	16 GB	14 174 624	8,066 505	120,477 247	2 031,115	135,992
	1 024	32 MB	32 GB	29 485 483	16,776 389	982,773 749	1 953,221	33,342
	8	ULONG_MAX	32 GB	3 969 183 958	17,642 095	283,746 065	1 857,376	115,484
C	1 024	16 kB	16 MB	10 884	0.007 010	0.024 739	2 282,454	646,752
	1 024	16 MB	16 GB	11 819 580	5,733 101	213,303 019	2 857,790	76,811
	1 024	32 MB	32 GB	23 865 849	11,573 146	509,545 807	2 831,382	64,308
	8	ULONG_MAX	32 GB	5 285 174 011	20,022 825	561,612 424	1 636,532	58,346

Tabuľka 5.4: Výsledky meraní funkcie CryptGenRandom

Špecifikácia [45]

Na inicializáciu potrebujeme štvoricu parametrov:

1. `BCRYPT_ALG_HANDLE hAlgorithm` – popis algoritmu CSP. Vytvára sa použitím funkcie `BCryptOpenAlgorithmProvider`. Tú však nie je nutné inicializovať. Pri použití makra `NULL}` sa použije predvolený poskytovateľ[8], ktorý poskytuje služby generovania náhodných čísel.
2. `PUCHAR pbBuffer` – adresa úložiska dát. Veľkosť musí byť najmenej cbBuffer.
3. `ULONG cbBuffer` – veľkosť vygenerovaných dát – maximálne `ULONG_MAX`.
4. `ULONG dwFlags` – značka na modifikovanie správania funkcie.

Posledný z parametrov môže nadobúdať hodnoty:

- nula – je nutné inicializovať CSP,
- `BCRYPT_RNG_USE_ENTROPY_IN_BUFFER`[9] – obsah dát v pbBuffer sa použije ako dodatočný zdroj entropie.
- `BCRYPT_USE_SYSTEM_PREFERRED_RNG`[10] – v prípade, že CSP je rovný NULL, volíme tento parameter.

BCryptGenRandom je typu `NTSTATUS`. Návratové hodnoty môžu byť `STATUS_SUCCESS`, `STATUS_INVALID_HANDLE` a `STATUS_INVALID_PARA-METER`.

[8] Microsoft Cryptographic Service Provider
[9] Od verzie Windows 8 a vyššie je ignorovaný.
[10] Windows Vista nepodporuje túto značku.

Zdrojový kód 5.4: Ukážka použitia BCryptGenRandom

```
#include<stdio.h>
#include<windows.h>
#include<ntstatus.h>
int main(){
  BYTE   *pbData=(BYTE*)malloc(sizeof(BYTE)* 10);
  if (STATUS_SUCCESS!=BCryptGenRandom(NULL,pbData,10,
                        BCRYPT_USE_SYSTEM_PREFERRED_RNG))
  printf("BCryptGenRandom error.\n");
  else    printf("Random sequence generated.\n");
  free(pbData);
  return 0;
}
```

Výsledky experimentálnych meraní

Výsledky meraní funkcie sú znázornené pomocou tabuľky 5.5

Počítač	Špecifikácie meraní							
	NI	BS	V_D	ANC	T_A [s]	T_B[s]	$\approx P_A$ [MB/s]	$\approx P_B$ [MB/s]
A	1 024	16 kB	16 MB	15 376	0,005 560	0,020 745	2 877,698	771,270
	1 024	16 MB	16 GB	16 905 128	5,981 131	236,919 392	2 739,281	69,154
	1 024	32 MB	32 GB	33 805 640	11,960 016	506,876 314	2 739,796	64,647
	8	ULONG_MAX	32 GB	4 538 998 240	12,544 902	568,900 696	2 612,057	57,599
B	1 024	16 kB	16 MB	11 011	0,007 250	0,066 015	2 206,897	242,369
	1 024	16 MB	16 GB	14 093 652	8,018 871	117,026 229	2 043,180	140,003
	1 024	32 MB	32 GB	29 396 766	16,724 702	845,170 274	1 959,257	38,771
	8	ULONG_MAX	32 GB	3 951 000 676	17,560 018	252,854 179	1 866,057	129,592
C	1024	16 kB	16 MB	10 912	0,005 963	0,025 895	2 683,213	617,880
	1 024	16 MB	16 GB	11 825 926	5,734 744	102,225 511	2 856,971	160,273
	1 024	32 MB	32 GB	23 915 195	11,596 222	437,033 046	2 825,748	74,978
	8	ULONG_MAX	32 GB	3 346 909 253	12,677 676	591,363 044	2 584,701	55,411

Tabuľka 5.5: Výsledky meraní funkcie BCryptGenRandom

5.3 Knižničné rozhrania

Používateľ pracujúci na platforme Windows môže okrem softvérových riešení použiť aj implementácie rôznych knižníc. Uvedenou metódou môžeme vytvoriť aplikáciu, resp. službu, nezávislú od operačného systému. Tzv. **muti-platformové programy**. V tejto kapitole demonštrujeme vyššie opísaný postup. Použijeme štan-

dardné knižničné rozhrania jazyka C a celosvetovo známu kryptografickú knižnicu – OpenSSL.

5.3.1 rand() a srand()

Rand je zrejme najznámejšou funkciou na generovanie pseudonáhodných výstupov, s ktorou sa používateľ stretne. Vo väčšine prípadov je implementáciou lineárne kongruentného generátora. Jeho inicializačná hodnota sa mení pomocou `srand()`. Pri tejto metóde je teda nutnosťou kooperácia týchto funkcií. Ak by sme neurčili seed generátora, výstup by bol vždy rovnaký[11]. Výstupom funkcie je hodnota z číselnej množiny prvkov 0 až 32767. Použitie rozhrania na kryptografické účely sa **neodporúča**. Z toho dôvodu nevykonáme testovanie tohto rozhrania. Príkladom použitia je zdrojový kód 5.5.

Zdrojový kód 5.5: Príklad použitia rand

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main () {
5    time_t t;
6    /* Intializes random number generator */
7    srand((unsigned) time(\&t));
8
9    /* Print random numbers from 0 to 32767 */
10   printf("%d\n", rand());
11   return(0);
12 }
```

Viac informácia čitateľ nájde v [46].

5.3.2 rand_s

`Rand_S`[12] je vylepšením rozhrania rand. Výstupom sú pseudonáhodné dáta v rozmedzí od 0 až `UINT_MAX`[13]. Funkcia používa operačný systém, aby vygenerovala kryptograficky bezpečné náhodné dáta. Nepotrebuje srand funkciu.

[11] Rovný srand(1)
[12] Kompatibilné iba s OS Windows
[13] UINT_MAX = ULONG_MAX = 4294967295

Špecifikácia

Funkcia používa jeden vstupný parameter. Ním je adresa na uloženie náhodnosti. Premenná musí byť typu `unsigned int`. `Rand_s` je typu `errno_t`. Návratové hodnoty sú nula pri úspechu alebo chybový kód pri zlyhaní. Príkladom použitia je zdrojový kód 5.6.

Zdrojový kód 5.6: Príklad použitia rand_s

```
1  #include <stdio.h>
2  #define _CRT_RAND_S
3  #include <stdlib.h>
4
5  int main () {
6    unsigned int data;
7    //generating 10*32bit value
8    for (int i = 0; i < 10 ; i++)
9    if(rand_s(&data)==0)
10     printf("%u\n",data);
11   else printf("Rand_S error\n");
12   return 0;
13 }
```

Pred použitím je nutné definovať makro `_CRT_RAND_S` pred linkovaním knižnice **stdlib.h**. Tým je zabezpečená deklarácia `rand_s`.

Pri hlbšom skúmaní funkcie sme zistili, že **používa rozhranie RtlGenRandom**. To je obsahom podkapitoly 5.2.1. Pri testovaní by teda generovanie trvalo dlhšie, pretože je nutné vykonať viacero krokov. Kvalita výstupných dát je rovnaká ako pri uvedenom rozhraní. Z týchto dôvodov sme v tejto práci neaplikovali testovacie metódy na rozhranie `rand_s`.

5.3.3 OpenSSL

OpenSSL je široko použiteľná a modifikovateľná knižnica, určená pre kryptografické aplikácie. Jej obsahom sú kvalitné a udržiavané algoritmy. Obdobne obsahuje aj funkcie, ktorých úlohou je generovanie kryptograficky bezpečných náhodných dát. Podľa dokumentácie [47] je ich implementácia skonštruovaná podľa odporúčania [2], teda knižnica **používa** na generovanie čísel **AES256-DRBG v počítadlovom režime**. Ekvivalentná bezpečnosť výstupov je teda na úrovni 256 bitov.

Špecifikácia [48, kap. 5]

V tejto práci budeme pracovať s verziou 1.1.1k[14]. Predvolený generátor sa inicializuje pri spustení a automaticky vykonáva reseed-ovanie. Pri tomto procese používa dôveryhodné zdroje daného operačného systému. V prípade OS Windows nimi sú výstupy funkcie `BCryptGenRandom` a `CryptGenRandom`. Tie sú popísané v podkapitole 5.2

Prístup k CSPRNG sprostredkúva dvojica funkcií:

- `int RAND\_bytes(unsigned char * buf, int num),`
- `int RAND\_priv\_bytes(unsigned char * buf, int num).`

Obidve sú deklarované v **rand.h** a poskytujú výstup z rovnakého CSPRNG. Druhá z uvedených funkcií používa unikátnu inštanciu generátora. Odporúča sa ju používať pri produkcii citlivých dát. Napríklad pri procese generovania kľúčov. Ukážku jednoduchého použitia v jazyku C znázorňuje zdrojový kód 5.7. Viac informácií o metódach RNG rozhraní je možne nájsť v riporte [48, kap. 5]

[14] Aktualizovaná 25.03.2021

Zdrojový kód 5.7: Príklad použitia funkcií OpenSSL

```
#include <stdio.h>
#include <openssl/rand.h>

int main(){
  unsigned char * data =(unsigned char*)malloc(
                          sizeof(unsigned char)*10);
  unsigned char * data2 =(unsigned char*)malloc(
                           sizeof(unsigned char)*10);

  if(RAND_bytes(data,10))
    for (int i = 0; i < 10; i++)
      printf("%u", data[i]);
  else printf("RAND_bytes error\n");
  printf("\n");
  if (RAND_priv_bytes(data2,10))
    for (int i = 0; i < 10; i++)
      printf("%u", data2[i]);
  else printf("RAND_priv_bytes error\n");

  free(data);
  free(data2);
  return 0;
}
```

Výsledky experimentálnych meraní

Aj napriek faktu, že knižnica čerpá zdroj náhodnosti z rozhraní operačného systému Windows, sme sa rozhodli aplikovať naše testovacie metódy na funkcie v OpenSSL. Dôvodom je vysoká miera používania v bežnej prevádzke. Prehľad nameraných hodnôt pri časovom meraní znázorňujú tabuľky č.:

- 5.6 – funkcia `RAND\_bytes`,
- 5.7 – funkcia `RAND\_priv\_bytes`.

Počítač	**Špecifikácie meraní**							
	NI	BS	V_D	ANC	T_A **[s]**	T_B**[s]**	$\approx P_A$ **[MB/s]**	$\approx P_B$ **[MB/s]**
A	1 024	16 kB	16 MB	12 700	0,008 139	0,022 769	1 965,843	702,710
	1024	16 MB	16 GB	12 371 900	7,039 314	114,814 303	2 327,500	142,700
	1 024	32 MB	32 GB	23 384 922	13,304 316	806,312 717	2 462,960	40,640
	16	INT_MAX	32 GB	1 662 721 364	9,190 874	814,555 646	3 565,276	40,228
B	1 024	16 kB	16 MB	12 700	0,008 139	0,022 769	1 965,843	702,710
	1 024	16 MB	16 GB	12 371 900	7,039 314	114,814 303	2 327,500	142,700
	1 024	32 MB	32 GB	23 384 922	13,304 316	806,312 717	2 462,960	40,639
	16	INT_MAX	32 GB	2 079 574 215	18,485 125	252,108 430	1 772,669	129,976
C	1 024	16 kB	16 MB	14 043	0,007 516	0,021 354	2 128,792	749, 274
	1 024	16 MB	16 GB	10 805 372	5,239 904	232,817 471	3 126,775	70,373
	1 024	32 MB	32 GB	21 915 623	10,626 684	517,460 051	3 083,558	63,325
	16	INT_MAX	32 GB	1 515 908 023	11,484 146	564,056 276	2 853,325	58,093

Tabuľka 5.6: Výsledky meraní funkcie RAND_bytes

Počítač	**Špecifikácie meraní**							
	NI	BS	V_D	ANC	T_A **[s]**	T_B**[s]**	$\approx P_A$ **[MB/s]**	$\approx P_B$ **[MB/s]**
A	1 024	16 kB	16 MB	16 877	0,006 082	0,005 921	2 630,714	2 702,246
	1 024	16 MB	16 GB	12 359 089	4,372 758	255,915 867	3 746,834	64,021
	1 024	32 MB	32 GB	24 664 863	8,726 003	650,850 067	3 755,213	50,346
	16	INT_MAX	32 GB	1 261 401 535	6,972 535	436,126 318	4 699,582	75,134
B	1 024	16 kB	16 MB	12 676	0,007 968	0,008 675	2 008,032	1 844,380
	1 024	16 MB	16 GB	12 450 851	7,084 268	116,442 791	2 312,730	140,704
	1024	32 MB	32 GB	23 206 632	13,202 893	1 006,623 945	2 481,880	32,552
	16	INT_MAX	32 GB	1 566 820 326	13,927 308	279,844 212	2 352,788	117,094
C	1 024	16 kB	16 MB	13 622	0,007 342	0,007 162	2 179,243	2 234,013
	1 024	16 MB	16 GB	10 483 219	5,083 700	217,651 658	3 222,849	75,276
	1 024	32 MB	32 GB	21 078 845	10,220 949	514,842 818	3 205,965	63,647
	16	INT_MAX	32 GB	2 431 597 459	18,421 180	613,641 577	1 778,822	53,399

Tabuľka 5.7: Výsledky merania funkcie RAND_priv_bytes

6 Bezpečnostné riziko pri RNG v prostredí VM

Virtuálny stroj (ďalej VM) je softvér, pomocou ktorého dokážeme vytvoriť abstraktné, respektíve virtuálne prostredie medzi našim počítačom a inou platformou. Celý tento proces je nezávislý od aktuálne použitého operačného systému (ďalej OS). Používateľ dokáže takto spustiť aj aplikačné sady, ktoré nie su určené práve pre jeho OS. VM umožní inštaláciu virtuálneho OS na aktuálnom systéme pomocou zdieľania hardvérového vybavenia počítača. V dnešnej dobe je virtualizácia vysoko populárna. Svoje uplatnenie našli v rôznych sférach. Príkladom použitia sú spoločnosti zamerané na virtualizáciu sieti. Prostredníctvom jednoduchej náhrady alebo rozšírenia softvéru dokážu poskytnúť väčšiu spoľahlivosť a flexibilitu svojich služieb, bez nutnosti nákupu viacerých zariadení. Obdobne zasahuje virtualizácia aj do procesu vývoja a testovania aplikácií.

Problematika VM je natoľko rozsiahla, že by mohla byť predmetom osobitnej práce. Avšak vzhľadom k aktuálnej problematike je opis základných princípov a metód týchto nástrojov vynechaný. Čitateľ má možnosť získať informácie prostredníctvom odkazu na video [49]. Autor v priebehu úvodných minút opisuje základné princípy virtualizácie. Následne aj demonštruje spustenie konkrétneho linuxového operačného systému.

V publikácií [50], autori zrealizovali úspešný útok vo virtuálnom prostredí s OS Windows. Dokument opisuje útok na implementáciu TLS protokolu pri štandardne zabezpečenej komunikácii medzi serverom a klientom. TLS (z ang. *Transport Secure Layer*) sa používa na zabezpečenie komunikácie s využitím certifikátov pri distribúcií verejného kľúča. Absolútnym základom útoku je vedomosť tzv. bezpečnostnej zraniteľnosti pri obnovení snímky obrazu (z ang. VM reset vulnerabilities). Následne autori dokázali pri opakovanom obnovení snímky extrahovať tajný kľúč servera. Ten vznikne deterministickým procesom pričom sa použijú náhodné dáta z generátorov náhodných čísel (z ang. Random Number Generator, ďalej RNG). Vlastník tohto údaju sa môže vydávať za server a klient

nedokáže rozoznať rozdiel.

Táto kapitola je opisom rovnakého bezpečnostného rizika avšak so zameraním na produkciu náhodných dát po obnovení snímky obrazu s OS Windows. Overenie aktuálnosti problému realizujeme experimentom. Použijú sa rozhrania systému Windows a nástroja OpenSSL, na generovanie kryptograficky bezpečných pseudo-náhodných čísel.

6.1 Opis bezpečnostného rizika

Pred opisom problému je nutné vysvetliť pojem snímka obrazu. Virtuálne stroje poskytujú možnosť spustiť ľubovoľný OS na počítači, bez nutnosti zmeny aktuálneho systému. Ďalšou z vymožeností tohto prostredia je možnosť vytvorenia kópie aktuálneho stavu pomocou tzv. snímky obrazu (z ang. *snapshots*). Pri tomto úkone dochádza k úplnému uloženiu stavu daného systému. Výsledkom je súbor, vďaka ktorému je možné kedykoľvek obnoviť systém do stavu, aký bol počas vytvorenia snímky. Vrátane všetkých údajov v pamäti. Využitie je výhodné najmä pri spúšťaní alebo inštalácií programov z neznámych zdrojov. Tie môžu poškodiť OS. Ďalším príkladom je aktualizácia systému. Po znehodnotení OS je takto možné obnoviť systém do bodu kedy bolo všetko v poriadku. Po načítaní snímky je možné ďalej pokračovať v práci avšak iba s dátami z obdobia vzniku snímky.

VM reset vulnerabilities pomenúva pôvodný anglický názov pre bezpečnostné riziko. Doslovný preklad do slovenčiny nie je celkom výstižný. Preto zavedieme pomenovanie problému ako – Zraniteľnosť pri obnovení snímky obrazu VM. Po reštarte uvedeným spôsobom, je možné zreprodukovať aj výstupy systémových CSPRNG. Táto skutočnosť rapídne znižuje bezpečnosť algoritmov ako AES [14, kap. 5], ktorý sa používa pri symetrickom šifrovaní, resp. dešifrovaní dát práve pomocou tajného kľúča. Ďalej DSA (z ang. *Digital Signature Algorithm*)[14, kap. 11], používaný pri digitálnych podpisoch. Ak má útočník k dispozícií kľúče, s ktorými uvedené nástroje pracujú, ich použitie za účelom zabezpečenia komunikácie je bezvýznamné. Pri frekventovanom používaní rovnakej snímky obrazu si teda útočník môže všimnúť podobnosti kľúčov.

6.2 Experimentálne overenie uvedenej zraniteľnosti

Realizáciu experimentu zabezpečí prenosný počítať A. Jeho konfigurácia je obsahom tabuľky 4.1. Pri tomto experimente je použitá iná metodika práce ako v

[50]. Vykonanie jedného z pokusov zabezpečí CSPRNG API operačného systému Windows 10. Konkrétne funkcia BCryptGenRandom [45]. Výstupy následne odhalia bezpečnostné riziko.

Prvé kroky sú spojené s voľbou hypervízora. Voľne dostupný nástroj je VirtualBox [51][1]. V čase experimentu je verziu 6.0.24. Nasleduje voľba OS. Použili sa 64-bitový systém. Konkrétne Windows 10 Pro vo verzii 1909 (zostava 18363.592). Konfigurácia systému prebieha podľa video návodu [52]. Dodatočne došlo k úprave možnosti zdieľania súborov na obojsmernú a navýšeniu počtu priradených procesorov (ďalej CPU) na počet 4. Dôvodom je samotný OS, ktorý sa v prípade použitej konfigurácie zdal pomalý na prácu dvoch CPU. Tieto zmeny však neovplyvnia výsledok experimentu.

Prostredie Guest VM

Po úspešnej inštalácii obrazu a sprístupnení OS je nutná príprava prostredia.

Vykonané inštalácie

- GCC prekladač – Winlibs GCC, 10.2.0, 64-bitová verzia, pre potreby prekladu programov do strojového kódu.[2]
- VirtualBox Extension Pack – 6.0.24, zabezpečí ovládače pre VM,
- Visual Studio Code – prostredie na úpravu, kompiláciu, a aj spustenie programov pomocou príkazového riadka.

6.2.1 Postup pri realizácií experimentu

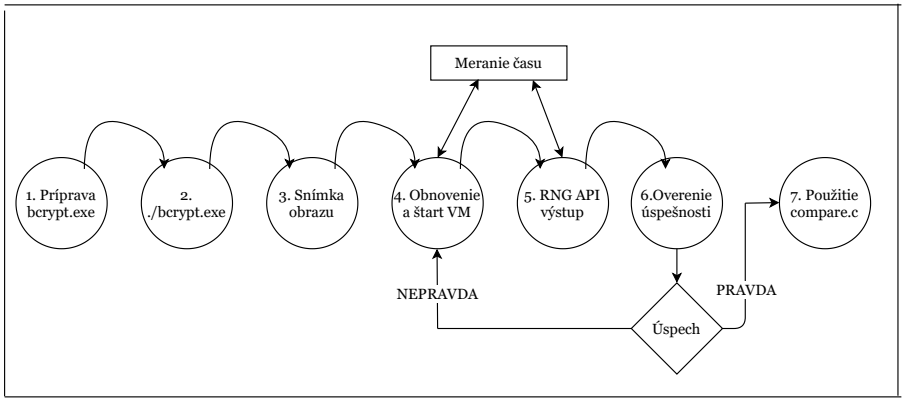
Pokus je síce realizovaný prostredníctvom odporúčaného rozhrania na generovanie náhodných čísel – BCryptGenRandom[45]. Avšak akékoľvek systémové RNG API dosahuje rovnaké výsledky.

Podstata nášho experimentu spočíva v presnom zopakovaní tých istých krokov od načítania snímky obrazu v približne rovnakom čase. Grafický postup znázorňuje schéma 6.1. V nasledujúcej časti je slovný opis.

Postup:

1. Príprava a preklad jednoduchého programu – bcrypt. Jeho úlohou je generovanie náhodných čísel pomocou systémového rozhrania Windows. Viď.

[1] Dostupný na: `https://www.virtualbox.org/wiki/Downloads`
[2] Dostupný na `https://winlibs.com/`



Obr. 6.1: Grafické znázornenie krokov postupu

ukážku zdrojového kódu tohto programu 6.1. Veľkosť výstupu vygenerovaných dát je v kóde stanovená na 1 920 B. Táto hodnota bola vykonaná pri poslednom experimente. Získaná je postupným navyšovaním od 15 bajtov. Teda 15 – 30 – 60 – ... – 1920.

Zdrojový kód 6.1: Ukážka zdrojového kódu bcrypt.c

```
#include<stdio.h>
#include<windows.h>
#include<ntstatus.h>
//gcc bcrypt.c -o bcrypt -lbcrypt
int main(){
  int size= 1920;//size of data to generate in bytes
  BYTE   pbData[size];
  BCryptGenRandom(NULL,pbData,size,
                  BCRYPT_USE_SYSTEM_PREFERRED_RNG);
  for( int i = 0; i < size;i++ )
    printf("%02X",pbData[i]); //02X for same printing
  printf("\n");
  return 0;
}
```

2. Po príprave algoritmu je nutné aby sa používateľ presunul do bodu, tesne pred spustením programu bcrypt, v príkazovom riadku. Pri experimente je použitý Windows PowerShell, prostredníctvom editora Visual Studio Code.

3. V uvedenom bode je nutné vytvoriť snímku obrazu. Operačný systém bežiaci vo virtuálnom prostredí je možné následne vypnúť.

4. V tomto bode používateľ potrebuje pre dosiahnutie úspešnosti používať

merací nastroj, napríklad stopky. Meranie je potrebné začať v ľahko identifikovateľnom mieste. Pri pokuse je aktivácia stopiek vykonaná v okamihu spustenia obnovenej snímky OS. Po sprístupnení systému sme spustili program. V čase jeho spustenie sme zastavili meranie času. Dôležité pre úspešný experiment je vždy sa snažiť trafiť do tohto času. Tento krok nie je celkom jednoduchý a vyžaduje si zručnosť používateľa.

Pri pokuse sme program spúšťali približne 2 sekundy po sprístupnení systému. V čase od začiatku načítavania obrazu až po zapnutie programu nebolo počas experimentu vykonané nič. Žiaden pohyb myši ani stlačenia klávesnice. Toto je aj dôvod, prečo je snímka obrazu zaznamenaná pred zapnutím programu bcrypt.

5. Takto vzniknuté dáta je nutné uložiť mimo VM. Dôvodom je, že pri obnovení pomocou snímky dôjde k zahodeniu všetkých dát, ktoré nie sú jej obsahom. Vrátane dát vzniknutých v kroku 4.

6. V prípade neúspešnej realizácií experimentu je potrebné opakovať tento proces od 4.kroku. Opakovanie je ukončené v prípade podozrenia, že daná sekvencia už bola vygenerovaná. Stačí ak sa používateľ zameria na prvé znaky výstupu.

7. Hneď po zaznamenaní zhody v počiatočných hodnotách podrobíme dáta testovaniu zhody. Za účelom tohto procesu je vytvorený program compare. Jeho úlohou je kontrola zhody v dvoch rovnako dlhých reťazcoch. Zdrojový kód je znázornený v ukážke 6.2. Program je nutné pri každej zmene kompilovať pomocou GCC prekladača. Za zhodu je považovaná rovnosť hodnoty a polohy vygenerovanej postupnosti. Výstup poskytuje hodnotu nájdenej zhody v percentách. V prípade úplnej zhody je používateľ upozornený slovným výstupom.

Zdrojový kód 6.2: Ukážka zdrojového kódu compare.c

```
#include <stdio.h>
#include <string.h>
//gcc compare.c -o compare
/*
place for generated data
*/
int main(){
  char string1[]={"1.dataToCompare"};
  char string2[]={"2.dataToCompare"};
  int size=sizeof(string1)/sizeof(string1[0]);
  if(strcmp(string1, string2)==0){
    printf("ABSOLUTE␣MATCH\n");
  }else {
    int i=0,counter=0;
    while(i<size){
      if(string1[i]==string2[i])
      counter++;
      i++;
    }
    printf("MATCH␣in␣%.2f␣percent\n",
    (double)(counter*100)/size);
  }
  return 0;
}
```

Experiment s knižnicou OpenSSL v.1.1.1k

Po úspešnom prvom experimente so systémovými rozhraniami je overená aj knižnica OpenSSL. Pri generovaní náhodných dát je použitý ako zdroj náhodnosti výstup z CSPRNG API systému, na ktorom je naimplementová zvolená funkcionalita. Teda ak je niektorá z funkcií na RNG implementovaná v OS Windows, tak vstupom OpenSSL generátora je výstup funkcie BCryptGenRandom. Periodicky dochádza k reseed procesu, pričom prvotná inicializácia prebehne pri spustení.

Pri experimente boli aplikované rovnaké metódy ako pri softvérových rozhraniach. Na generovanie bol použitý príkaz, s ktorým sa stretneme hlavne pri RSA [14, kap. 7.5.3] algoritme:

```
openssl genrsa -out myCA.key 2048
```

Príkaz vytvorí súkromný kľúč s veľkosťou 2048 bajtov. Ten je možné uplatniť v TLS protokole, konkrétne pri distribúcií kľúčov s využitím certifikátov.

6.3 Vyhodnotenie výsledkov testovania

Vzhľadom k dobe, ktorá ubehla od uverejnenia článku [50], bolo očakávané úplné vyriešenie tohto bezpečnostného rizika. Počas experimentu sa podarilo vyvrátiť túto hypotézu. Dôkazom je videozáznam, ktorý demonštruje postup od 4. do 7 kroku experimentu. Je obsahom prenosného média prílohy A. Zároveň je možné ich vyhľadať na webovej platforme Youtube [53]. Vďaka videu je zrejmé, že isté bezpečnostné riziko pretrváva. Obsahom uvedenej prílohy A sú aj použité zdrojové kódy s balíčkom makefile.

Obdobne sme overili aj vytváranie súkromného kľúča pomocou OpenSSL. Používatelia tejto knižnice nemusia byť veľmi znepokojený, pretože sa nám opísaným postupom nikdy nepodarilo vytvoriť rovnaký kľúč. Avšak maximálna dosiahnutá zhoda pomocou programu compare, bola 5,43%. Táto hodnota zhody už sama o sebe nie je typická pre CSPRNG. Dôvodom je aktualizácia vnútorného stavu generátora pri každej inicializácií tejto knižnice. Tým sa zabezpečí pseudonáhodnosť aj v prípade, že bola použitá rovnaká entropia systému. Je potrebné si uvedomiť, že bezpečnosť takto vygenerovanej postupnosti je omnoho nižšia ako pri bežnom pracovnom postupe rozhrania.

Oplatí sa poukázať na skutočnosť, že zopakovať výstup generovania s rovnakým výsledkom si vyžaduje splnenie podmienok, ktoré užívateľ pri bežnej práci nevykonáva. Viď vyššie uvedený postup. Obdobne bolo potrebné počas testovania vykonať veľké množstvo pokusov. Niekedy bolo nutné počas generovaní konkrétnej sekvencie absolvovať reštart obrazu aj 20-krát.

V porovnaní s [50], je situácia v dnešnej dobe pravdepodobne nenapodobiteľná. Je nutné si uvedomiť, že istý potenciál na útočenie existuje. Ak Windows 10 beží prostredníctvom hypervízora, tzv. Hyper-V, tak po obnovení následne dochádza k obnoveniu vnútorných inicializačných hodnôt (z ang. reseed) stavov hlavného generátora systému. Týmto sa značne redukuje čas, kedy je možné uvedený bezpečnostný problém využiť. Jeho implementácia je dnes štandardná. Okrem toho väčšina softvérových hypervízorov pracuje aj v kooperácií s hardvérovými ako Intel VM-x a AMD-V. Jedinou podmienkou je ich povolenie v BIOS-e. Na dnešných počítačoch je však možnosť virtuálizácie hardvéru štandardne zapnutá. Používateľ teda nie je nútený zasahovať do týchto nastavení.

Aktualizácia softvérového vybavenia

V čase tvorby práce došlo k aktualizáciám softvérov použitých v tejto kapitole. Zmeny sa týkajú:

1. aktualizácia OS Windows – 10, Pro, 64-bitová verzia, v20H2, 19042.985,
2. GCC prekladač – Winlibs, 64-bitový, v11.1.0,
3. VirtualBox – 6.1.22 (vrátane doplnkov).

Testovanie bolo vykonané aj pomocou aktualizovaných nástrojov. Bezpečnostné riziko aj napriek uvedeným zmenám stále pretrváva. Aktualizovaný videozáznam ([54]) spoločne s údajmi o nástrojoch je taktiež obsahom prílohy A.

Je nutné informovať čitateľa o možnom probléme pri realizácii vyššie uvedených skutočností. Ak je na natívnom OS Windows použitý Windows Subsystem pre Linux vo verzii 2 (WSL2), tak je vysoko pravdepodobné, že nedosiahnete výsledky uvedené v tejto publikácií. Pred aplikovaním metód je potrebné deaktivovať túto funkciu. Dôvodom je interferencia medzi Hyper-V a týmto systémom. Občasne môže spôsobovať aj pády, respektíve nespoľahlivý výkon systému.

Kapitola demonštruje aktuálny bezpečnostný problém v prostredí OS Windows na platforme VM. Teoreticky je teda možné použiť uvedenú chybu aj pri pokuse o sieťový útok. Tento problém je dlhodobo známy fakt. Vzhľadom k veľkej popularite a rozvoju VM očakávame v blízkej dobe úplne odstránenie uvedeného bezpečnostného rizika.

7 Vyhodnotenie dosiahnutých výsledkov

Obsahom tejto kapitoly je vyhodnotenie dosiahnutých výsledkov, ktoré sme získali počas experimentálnych meraní a štatistického testovania náhodných dát. Následne pomocou týchto dát vytvoríme odporúčanie pre použitie funkcií v prostredí OS Windows.

Vyhodnotenie štatistického testovania dát

V tabuľke 7.1 je výsledné poradie určené zostupným usporiadaním podľa kvality funkcií. Pre účely usporiadania sme vykonali súčet p-hodnôt každého testu z výstupu sady NIST STS v súboroch FinalAnalysisReport.txt[1]. Tieto dáta pre konkrétne funkcie sú obsahom prílohy A. Zároveň sme vypočítali celkovú úspešnosť na základe pomeru počtu úspešných a všetkých testov. Pomocou týchto dvoch hodnôt sme následne určili výsledné poradie uvedenej tabuľky, pričom prednosť dostali funkcie s väčšou úspešnosťou.

Pri spustení štatistickej sady na každý výstup náhodných dát testovanej funkcie sme použili rovnaké inicializačné parametre. Konkrétne, vstupná sekvencia – 1 000 000 bitov, počet prúdov – 4600 a pred definované hodnoty testov nezmenené. Pri uvedenej konfigurácii sady trvalo jedno testovanie okolo 10 hodín. Testy boli vykonané na počítači A. Špecifikácia tohto zariadenia je obsahom tabuľky 4.1.

Vyhodnotenie experimentálnych meraní

Na základe označenia prvkov 4. a priebežných experimentálnych meraní v 5. kapitole, je vytvorená tabuľka 7.2.

Poradie rozhraní v uvedenej tabuľke vzniklo nasledujúcim spôsobom. Vypočítali sme súčet ANC všetkých počítačov pri maximálnej možnej veľkosti BS. Takto

[1] Použili sme nástroj Magma. Dostupný na: http://magma.maths.usyd.edu.au/calc/

Funkcia	Hodnoty					
	Poradie	API	Testy celkovo	Úspešné testy	Úspešnosť [%]	$\approx \sum_{i=1}^{188} p_i$
RAND_priv_bytes	1	OpenSSL	188	188	100	98,348
RtlGenRandom	2	Windows	188	188	100	93,108
RAND_bytes	3	OpenSSL	188	188	100	92,394
RDSEED	4	AMD	188	188	100	89,538
BCryptGenRandom	5	Windows	188	187	99,47	91,298
CryptGenRandom	6	Windows	188	186	98,94	89,566
RDRAND	7	AMD	188	186	98,94	87,767

Tabuľka 7.1: Vyhodnotenie kvality výstupov funkcií pomocou NIST STS

vzniknuté číslo sme následne vydelili počtom meracích zariadení. Hodnoty jednotlivých rozhraní sme usporiadali vzostupne.

Funkcia	Hodnoty				
	Poradie	API	NI	BS	$\approx$ Priemer cyklov
RAND_bytes	1	OpenSSL	16	32 GB	$1{,}752 * 10^9$
RAND_priv_bytes	2	OpenSSL	16	32 GB	$1{,}753 * 10^9$
RtlGenRandom	3	Windows	8	32 GB	$3{,}925 * 10^9$
BCryptGenRandom	4	Windows	8	32 GB	$3{,}945 * 10^9$
CryptGenRandom	5	Windows	8	32 GB	$4{,}593 * 10^9$
RDRAND	6	AMD	8	32 GB	$1{,}445 * 10^{12}$
RDSEED	7	AMD	8	32 GB	$3{,}098 * 10^{12}$

Tabuľka 7.2: Vyhodnotenie funkcií podľa výsledkov meraní

Celkové vyhodnotenie získaných výsledkov

Na základe tabuliek 7.1 a 7.2 je očividné, že knižnica OpenSSL ponúka kvalitne spracované rozhranie na generovanie kryptograficky bezpečných náhodných čísel. Dominuje vo všetkých experimentoch, ktoré sú obsahom tejto práce. Ďalšími plusmi sú pravidelné aktualizácie, ľahká implementácia, možnosť použitia na rôznych typoch OS a voľná dostupnosť pre používateľa. Na základe uvedených faktov odporúčame používateľovi pri tvorbe kryptografický aplikácií na platforme Windows, použiť práve túto knižnicu. Pri kryptografických aplikáciach je výhodnejšie implementovať RAND_priv_bytes, pretože poskytla lepšie výsledky pri testovaní kvality ako funkcia RAND_bytes. Toto odporúčanie dostala obdobne aj od vývojárov knižnice.

V prípade systémových rozhraní dosiahla najlepšie výsledky funkcia RtlGenRandom. Problémom však ostáva jej podpora systému v budúcnosti. Prí tvorbe kryptografickej aplikácie je nevyhnutne nutné ošetriť kompatibilitu systému pomocou funkcie BCryptGenRandom. V prostredí virtuálnych strojov s OS Windows však jej použitie neodporúčame vzhľadom k existujúcemu bezpečnostnému problému.

Používateľ má k dispozícií ešte procesorové inštrukcie RDSEED a RDRAND. Obidve poskytujú dostatočne kvalitné dáta. Najmä prvá z uvedených. Ich najväčšou nevýhodou je čas produkcie náhodných dát. Ak pri implementácií nie je proces náchylný na rýchlosť vykonávania tak odporúčame použitie funkcie RDSEED.

8 Záver

Cieľom tejto práce bolo postupné uvedenie čitateľa do problematiky generovania náhodných čísel so zameraním na platformu Windows. Najprv sme vysvetlili základné pojmy súvisiace s témou práce. Následne sme pomocou systematickej klasifikácie charakterizovali nástroje na tvorbu náhodných dát, ktoré sú bežne využité v oblasti informačnej bezpečnosti. Zároveň sme vytýčili, ktoré z uvedených generátorov sú obsahom práce.

V úvode druhej kapitoly sme stručne opísali základné parametre operačných systémov. Vysvetlili sme dôležitosť kryptografických modulov a kvality výstupov generátorov pri bežnej prevádzke počítača. Následne sme charakterizovali historický vývoj systému Windows z hľadiska kryptografického vývinu. Vysvetlili sme základné princípy aplikované v prvom rozhraní – CAPI. Podrobnejší opis použitých metód sme však vykonali pri aktuálnom CNG API. So zameraním na tému tejto práce sme charakterizovali základné prvky systému. Opísali sme možnosti prístupu k rozhraniam. Najprv sme vysvetlili základné pojmy súvisiace s témou práce. Definovali sme čerpanie náhodnosti pri štarte a proces obnovovania počiatočnej inicializačnej hodnoty generátorov pomocou slova reseed.

Od praktík aplikovaných v kryptografických moduloch platformy Windows sme sa presunuli k testovaniu generátorov. Načrtli sme možnosti overovania výstupov pomocou sád štatistických testov. Následne sme opísali metódy vyhodnocovania v štatistickej testovacej sade NIST STS. Špecifikovali sme testy v súprave spoločne s meraním času potrebného na ich vykonanie. Pomocou uvedenej zbierky sme vykonali kontrolu kvality výstupných dát z RNG funkcií. Charakterizovali sme metódy, ktorými realizujeme experimenty na rozhraniach. Vytvorili sme označenia kvôli prehľadnejšej reprezentácii výsledkov a určili odchýlku meraní.

V piatej kapitole sme prešli k popisu možností generovania náhodných čísel. Popísali sme hárdverové a softvérové možnosti, ktoré má používateľ k dispozícii. Počas vykonávania jednotlivých funkcií sme aplikovali meracie metódy a výsledky reprezentovali vo forme tabuliek. V rámci overenia bezpečnosti ge-

nerovania v prostredí virtuálnych stojov sme vykonali experiment. Špecifikácia problému, realizácia pokusu a následné zhodnotenie sú obsahom predposlednej kapitoly.

V poslednej časti tejto práce sme vyhodnotili naše experimentálne merania spoločne s výsledkami štatistických testov. Následne sme vo forme odporúčania určili funkciu/e, ktorú by mal čitateľ pri implementácií kryptografických aplikácii na platforme Windows použiť.

Pre účely rozšírenia tejto práce by bolo vhodné aplikovať zvolené merania viac z vnútra operačného systému. Konkrétne implementáciou kernel modulov a následným experimentom. V ďalšej fáze definovať presné správanie rozhraní pri ich volaní. Tieto údaje následne schematicky načrtnúť a opísať. Uvedený postup sa dá rozšíriť na viacero systémových modulov. V prípade zlepšenia aktuálnej epidemiologickej situácie vykonať štatistické testovanie s viacerými testovacími sadami a väčším množstvom prúdov.

Literatúra

1. L'ECUYER, Pierre. Uniform random number generation. *Annals of Operations Research*. 1994, roč. 53, č. 1, s. 77–120. Dostupné tiež z: http://www.iro.umontreal.ca/~lecuyer/myftp/papers/tutaor-1994.pdf. [Online; Citované: 12.3.2021].
2. BARKER, Elaine; KELSEY, John; STANDARDS, National Institute of; TECHNOLOGY; COMMERCE, U.S. Department of. *NIST Special Publication 800-90A: Recommendation for Random Number Generation Using Deterministic Random Bit Generators*. North Charleston, SC, USA: CreateSpace Independent Publishing Platform, 2012. ISBN 1478169311. Dostupné tiež z: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.228.2294. [Online; Citované: 12.3.2021].
3. HASINOFF, Samuel W. *Photon, Poisson Noise.* 2014. Dostupné tiež z: https://people.csail.mit.edu/hasinoff/pubs/hasinoff-photon-2012-preprint.pdf/. [Online; 6.3.2021].
4. QUANTIQUE, Company ID. *What is Q in the QRNG - Random number generation White Paper*. 1227 Carouge/Geneva, Switzerland, 2020. Dostupné tiež z: https://marketing.idquantique.com/acton/attachment/11868/f-0226/1/-/-/-/-/What%20is%20the%20Q%20in%20QRNG_White%20Paper.pdf/. [Online; 6.3.2021].
5. MYRVOLD, Wayne; GENOVESE, Marco; SHIMONY, Abner. Bell's Theorem. In: ZALTA, Edward N. (ed.). *The Stanford Encyclopedia of Philosophy*. Fall 2020. Metaphysics Research Lab, Stanford University, 2020. Dostupné tiež z: https://plato.stanford.edu/archives/fall2020/entries/bell-theorem/. [Online; 6.3.2021].
6. WATT, AD; MAXWELL, EL. Characteristics of atmospheric noise from 1 to 100 kc. *Proceedings of the IRE*. 1957, roč. 45, č. 6, s. 787–794. Dostupné tiež z: https://ieeexplore.ieee.org/abstract/document/4056603. [Online; 6.3.2021].

7. MCINTYRE, R. J. Multiplication noise in uniform avalanche diodes. *IEEE Transactions on Electron Devices*. 1966, roč. ED-13, č. 1, s. 164–168. Dostupné z DOI: 10.1109/T-ED.1966.15651.

8. MAROUANI, Hicham; DAGENAIS, Michel R. Internal clock drift estimation in computer clusters. *Journal of Computer Systems, Networks, and Communications*. 2008, roč. 2008. Dostupné tiež z: https://www.hindawi.com/journals/jcnc/2008/583162/. [Online; 6.3.2021].

9. BARKER, Elaine; DANG, Quynh. Nist special publication 800-57 part 1, revision 5:Recommendation for key management: Part 1–general. *NIST, Tech. Rep*. 2020. Dostupné tiež z: https://blkcipher.pl/assets/pdfs/NIST.SP.800-57pt1r5.pdf. [Online; Citované: 12.3.2021].

10. CHEON, Audrey Chaeyoung. The Influence of Pseudorandom Number Generators. *ANALYSIS OF APPLIED MATHEMATICS*. 2017, s. 80. Dostupné tiež z: http://www.analysisofappliedmathematics.org/wp-content/uploads/2016/09/AAM2.pdf#page=80. [Online; Citované: 12.3.2021].

11. SCHRIFT, Avital W; SHAMIR, Adi. On the universality of the next bit test. In: *Conference on the Theory and Application of Cryptography*. 1990, s. 394–408. Dostupné tiež z: https://link.springer.com/content/pdf/10.1007/3-540-38424-3_29.pdf. [Online; Citované: 12.3.2021].

12. BARKER, Elaine. Recommendation for Digital Signature Timeliness. *NIST Special Publication*. 2009, roč. 800, s. 6. Dostupné tiež z: https://csrc.nist.rip/library/NIST%20SP%20800-102%20Recommendation%20for%20Digital%20Signature%20Timeliness,%202009-09.pdf. [Online; Citované: 12.3.2021].

13. DANG, Quynh; WOLFF, Otto J; CHANG, Shu-jen; DODSON, Donna F; KELSEY, John; PERLNER, Ray; POLK, W Timothy. NIST Special Publication 800-106 Randomized Hashing for Digital Signatures. 2009. Dostupné tiež z: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.153.2967. [Online; Citované: 12.3.2021].

14. LEVICKÝ, Dušan. *Kryptografia v informačnej bezpečnosti*. Elfa, 2005.

15. DORRENDORF, Leo; GUTTERMAN, Zvi; PINKAS, Benny. Cryptanalysis of the random number generator of the windows operating system. *ACM Transactions on Information and System Security (TISSEC)*. 2009, roč. 13, č. 1, s. 16–17. Dostupné tiež z: https://citeseerx.ist.psu.edu/viewdoc/

download?doi=10.1.1.481.1401&rep=rep1&type=pdf. [Online; Citované: 12.3.2021].

16. MCEVOY, Robert; CURRAN, James; COTTER, Paul; MURPHY, Colin. Fortuna: cryptographically secure pseudo-random number generation in software and hardware. 2006.

17. JEEVA, AL; PALANISAMY, Dr V; KANAGARAM, K. Comparative analysis of performance efficiency and security measures of some encryption algorithms. *International Journal of Engineering Research and Applications (IJERA)*. 2012, roč. 2, č. 3, s. 3033–3037. Dostupné tiež z: https://d1wqtxts1xzle7.cloudfront.net/28320314/SG2330333037-with-cover-page.pdf?Expires=1621881470&Signature=J80bk6ZBcDIdwBshWwXYaIokl7LXA6UocqttV5n6yKOYRj-3yhWHXWAdimDAPYMPoElfqSJWb2qaY1HlfSoEvmH6zNJb-ffXEu6ZIM4a94MppR~wTaRmSjngcV-ee6uOnvCDAS7mrjQREJjnfTEmaCqOBfk9gYewMpsHB4kdfzvemzIdo6aifC19JXpdFjLELIifUu6F6pyFC6AGDE-L3aNxKNRD8t8AH73NEk6VWdb9UvSJdAveQghaGJlcH3bb8-9lUGuTE4Aq-qI1CuGefflMpKZV-SVEcRJOQj23S8ZIv9WnNyCJwc8V3qGgwszYLezA6K~gyjtWCi5fYA3eHA__&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA. [Online; Citované: 12.3.2021].

18. FIPS, PUB. 197: Advanced encryption standard (AES). *National Institute of Standards and Technology*. 2001, roč. 26.

19. STALLINGS, William. *Operating systems: internals and design principles*. Boston: Prentice Hall, 2012. Dostupné tiež z: https://repository.dinus.ac.id/docs/ajar/Operating_System.pdf. [Online; Citované: 12.3.2021].

20. SILBERSCHATZ, Abraham; PETERSON, James L; GALVIN, Peter B. *Operating system concepts*. Addison-Wesley Longman Publishing Co., Inc., 1991. Dostupné tiež z: http://www.uobabylon.edu.iq/download/M.S%202013-2014/Operating_System_Concepts,_8th_Edition%5BA4%5D.pdf. [Online; Citované: 12.3.2021].

21. LIN, Chu-Hsing; YEH, Yi-Shiung; CHIEN, Shih-Pei; LEE, Chen-Yu; CHIEN, Hung-Sheng. Generalized secure hash algorithm: SHA-X. In: *2011 IEEE EUROCON-International Conference on Computer as a Tool*. 2011, s. 1–4. Dostupné tiež z: https://www.researchgate.net/profile/Chu-Hsing-Lin/publication/221290910_Generalized_secure_hash_algorithm_SHA-X/links/0c96052c81c724ce6a000000/Generalized-secure-hash-algorithm-SHA-X.pdf. [Online; Citované: 12.3.2021].

22. BASSHAM III, Lawrence E; RUKHIN, Andrew L; SOTO, Juan; NECHVATAL, James R; SMID, Miles E; BARKER, Elaine B; LEIGH, Stefan D; LEVENSON, Mark; VANGEL, Mark; BANKS, David L et al. *Sp 800-22 rev. 1a. A statistical test suite for random and pseudorandom number generators for cryptographic applications*. National Institute of Standards & Technology, 2010. Dostupné tiež z: `https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-22r1a.pdf`. [Online; Citované: 12.3.2021].

23. BARKER, Elaine; ROGINSKY, Allen; BLANK, Rebecca; GALLAGHER, Patrick D.; SECRETARY, Under. *NIST Special Publication 800-133 Recommendation for Cryptographic Key Generation*. 2012. Dostupné tiež z: `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.252.3846&rep=rep1&type=pdf`. [Online; Citované: 12.3.2021].

24. Entropy -infromation theory. 2021. Dostupné tiež z: `https://en.wikipedia.org/wiki/Entropy_(information_theory)`. [Online; Citované: 12.3.2021].

25. DORRENDORF, Leo; GUTTERMAN, Zvi; PINKAS, Benny. Cryptanalysis of the random number generator of the windows operating system. *ACM Transactions on Information and System Security (TISSEC)*. 2007, roč. 13, č. 1, s. 1–32. Dostupné tiež z: `https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.481.1401&rep=rep1&type=pdf`. [Online; Citované: 12.3.2021].

26. BROWN, Karen H. Security requirements for cryptographic modules. *Fed. Inf. Process. Stand. Publ*. 1994, s. 1–53. Dostupné tiež z: `http://mayor.fri.uniza.sk/krypto/04/fips140-2.pdf`. [Online; Citované: 12.3.2021].

27. ALZHRANI, Khudran; ALJAEDI, Amer. Windows and linux random number generation process: A comparative analysis. *International Journal of Computer Applications*. 2015, roč. 113, č. 8. Dostupné tiež z: `https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.695.7729&rep=rep1&type=pdf`. [Online; Citované: 12.3.2021].

28. FERGUSON, Niels. Going in-depth on the Windows 10 random number generation infrastructure. 2019, s. 1–19. Dostupné tiež z: `https://aka.ms/win10rng`. [Online; Citované: 12.3.2021].

29. Trusted Platform Module. 2021. Dostupné tiež z: `https://en.wikipedia.org/wiki/Trusted_Platform_Module`. [Online; Citované: 12.3.2021].

30. Advanced Configuration and Plan Interface. 2021. Dostupné tiež z: https://en.wikipedia.org/wiki/Advanced_Configuration_and_Power_Interface. [Online; Citované: 12.3.2021].

31. Unified Extensible Firmware Interface. 2021. Dostupné tiež z: https://en.wikipedia.org/wiki/Unified_Extensible_Firmware_Interface. [Online; Citované: 12.3.2021].

32. BROWN, Robert G. Dieharder: A Random Number Test Suite. 2003. Dostupné tiež z: https://webhome.phy.duke.edu/~rgb/General/dieharder.php. [Online; Citované: 11.5.2021].

33. SÝS, Marek; ŘÍHA, Zdeněk; MATYÁŠ, Vashek. Algorithm 970: optimizing the NIST statistical test suite and the berlekamp-massey algorithm. *ACM Transactions on Mathematical Software (TOMS)*. 2016, roč. 43, č. 3, s. 1–11. Dostupné tiež z: https://dl.acm.org/doi/abs/10.1145/2988228. [Online; Citované: 11.5.2021].

34. RÁČEK, Ing. Tomáš. *Prahovací pravidla pro potlačovční šumu ve zvukových signálech*. 2010. Dostupné tiež z: https://www.vutbr.cz/studenti/zav-prace?zp_id=32092. [Online; Citované: 12.5.2021].

35. QueryPerformanceCounter function (profileapi.h). 2020. Dostupné tiež z: https://docs.microsoft.com/en-us/windows/win32/api/profileapi/nf-profileapi-queryperformancecounter. [Online; Citované: 15.5.2021].

36. QueryPerformanceFrequency function (profileapi.h). 2020. Dostupné tiež z: https://docs.microsoft.com/en-us/windows/win32/api/profileapi/nf-profileapi-queryperformancefrequency. [Online; Citované: 15.5.2021].

37. Acquiring high-resolution time stamps. 2018. Dostupné tiež z: https://docs.microsoft.com/en-us/windows/win32/sysinfo/acquiring-high-resolution-time-stamps. [Online; Citované: 12.5.2021].

38. HAMBURG, Mike; KOCHER, Paul; MARSON, Mark E. *Analysis of Intel's Ivy Bridge digital random number generator*. 2012. Tech. spr. Technical Report, Cryptography Research INC. Dostupné tiež z: https://cdn.atraining.ru/docs/Intel_TRNG_Report_20120312.pdf. [Online; Citované: 12.3.2021].

39. FOG., Agner. *Instruction tables*. 2021. Dostupné tiež z: https://www.agner.org/optimize/instruction_tables.pdf. [Online; Citované: 16.5.2021].

40. MECHALAS, John P. Intel Digital Random Number Generator (DRNG) Software Implementation Guide. *Intel Corporation*. 2014. Dostupné tiež z: https://software.intel.com/content/www/us/en/develop/articles/intel-digital-random-number-generator-drng-software-implementation-guide.html. [Online; Citované: 12.5.2021].

41. PAOLONI, Gabriele. How to benchmark code execution times on Intel IA-32 and IA-64 instruction set architectures. *Intel Corporation*. 2010, roč. 123. Dostupné tiež z: https://www.intel.com/content/dam/www/public/us/en/documents/white-papers/ia-32-ia-64-benchmark-code-execution-paper.pdf. [Online; Citované: 12.5.2021].

42. AMD. AMD Secure Random Number Generator Library. 2019, s. 1–10. Dostupné tiež z: https://developer.amd.com/wp-content/resources/AMD%20Secure%20Random%20Number%20Generator%20Library%202.0%20-Whitepaper.pdf. [Online; Citované: 12.5.2021].

43. RtlGenRandom function (ntsecapi.h). 2018. Dostupné tiež z: https://docs.microsoft.com/en-us/windows/win32/api/ntsecapi/nf-ntsecapi-rtlgenrandom. [Online; Citované: 12.5.2021].

44. CryptGenRandom function (wincrypt.h). 2018. Dostupné tiež z: https://docs.microsoft.com/en-us/windows/win32/api/wincrypt/nf-wincrypt-cryptgenrandom. [Online; Citované: 15.5.2021].

45. BCryptGenRandom function (bcrypt.h). 2018. Dostupné tiež z: https://docs.microsoft.com/en-us/windows/win32/api/wincrypt/nf-wincrypt-cryptgenrandom. [Online; Citované: 15.5.2021].

46. ISO C Random Number Functions. [B.r.]. Dostupné tiež z: https://www.gnu.org/software/libc/manual/html_node/ISO-Random.html. [Online; Citované: 15.5.2021].

47. RAND - the OpenSSL random generator. [B.r.]. Dostupné tiež z: https://www.openssl.org/docs/man1.1.1/man7/RAND.html. [Online; Citované: 15.5.2021].

48. SAS, Quarkslab. OpenSSL Security Assessment – Technical Report – Ref. 18-04-720-REP. 2019, s. 1–35. Dostupné tiež z: https://eprint.iacr.org/2016/367.pdf. [Online; Citované: 15.5.2021].

49. NETWORKCHUCK. You need to learn Virtual Machines RIGHT NOW!! (Kali Linux VM, Ubuntu, Windows). 2021. Dostupné tiež z: `https://www.youtube.com/watch?v=wX75Z-4MEoM%5C&ab%5C_channel=NetworkChuck`. [Citované: 23.5.2021].

50. RISTENPART, Thomas; YILEK, Scott. When Good Randomness Goes Bad: Virtual Machine Reset Vulnerabilities and Hedging Deployed Cryptography. In: *NDSS*. 2010. Dostupné tiež z: `http://pages.cs.wisc.edu/~rist/papers/sslhedge.pdf`. [Online; Citované: 16.5.2021].

51. Oracle VM Virtual Box. [B.r.]. Dostupné tiež z: `https://en.wikipedia.org/wiki/VirtualBox`. 21.5.2021.

52. MIKETHETECH. Installing Windows 10 on Virtualbox 6.1.12 – FULL PROCESS, 2020 – video tutorial. 2020. Dostupné tiež z: `https://www.youtube.com/watch?v=gKQvaPejxpc&ab_channel=MikeTheTech`. [Online; Citované: 17.5.2021].

53. ROHAČ, Marek. Proof for security problem in Windows 10 v.1909 VM enviroment. 2021. Dostupné tiež z: `https://youtu.be/NTmtWbgm0gw`. [Online; Citované: 17.5.2021].

54. ROHAČ, Marek. Proof for security problem in Windows 10 VM enviroment – Update for version 20H2. 2021. Dostupné tiež z: `https://youtu.be/eaPwpNJcQr0`. [Online; Citované: 28.5.2021].

Zoznam príloh

Príloha A CD médium – viď. obsah CD média

A Obsah CD Média

Obsah tohto média je dostupný na gite:

- `https://git.kpi.fei.tuke.sk/marek.rohac/bc`
- `https://git.kemt.fei.tuke.sk/mr171hg/BachelorWork`

https://git.kemt.fei.tuke.sk/mr171hg/BachelorWork

```
bc-master/ ..... Koreňový priečinok
└── appendixes/ ..... Prílohy
    └── BC_ZK/ ..... Zdrojové kody
        ├── amdINC/ ..... Knižnice pre AMD API
        │   ├── secrng.c
        │   └── secrng.h
        ├── openssl/ ..... Knižnice pre OpenSSL
        │   ├── include/
        │   │   └── openssl/
        │   └── lib/
        │       ├── pkgconfig/
        │       ├── libcrypto.a
        │       ├── libcrypto.dll.a
        │       ├── libssl.a
        │       └── libssl.dll.a
        ├── Results/ ..... Výsledky experimentov
        │   ├── Measurment_Results/ ..... Výsledky meraní
        │   │   ├── PCA/ ..... Výsledky kofigurácie A
        │   │   ├── PCB/ ..... Výsledky kofigurácie B
        │   │   └── PCC/ ..... Výsledky kofigurácie C
        │   └── NIST_STS_Results/ ..... Výsledky štatistického testovania
        │       ├── bcryptgenrandom/
        │       ├── cryptgenrandom/
        │       ├── openssl/
        │       ├── rd_instructions/
        │       └── rtlgenrandom/
        └── testingPrograms/ ..... Programy použité pre meranie
            └── MEASURMENT/
                ├── bcryptgenrandom/
                └── cryptgenrandom/
```

- openssl/
- rdinstructions/
- rtlgenrandom/
- README.md
- runall.bat ..... Skript pre automatizované spustenie
- Modified_NIST_STS/ ..... Upravená sada NIST
  - experiments/
  - include/
  - obj/
  - src/
  - templates/
  - assess.exe ..... Spustiteľný program pre NIST STS
  - makefile
- VM_Experimnt/ ..... Dokumentácia k experimentu vo VM
  - source_codes/ ..... Použité zdrojové kódy
    - bcrypt.c
    - compare.c
    - makefile
  - Proof_OSv1909.mkv
  - Proof_UpdatedOS_v20H2.mkv
  - README.txt
- .gitignore
- amdSPRNG.c ..... Implementácia AMD API
- libcrypto-1_1-x64.dll
- makefile
- openssl_rng.c ..... Implementácia OpenSSL API
- README.md ..... Dokumentácia k zdrojovým kódom
- winAPIprng.c ..... Implementácia Windows API
- prilohaa.tex ..... Zdrojový .tex prílohy A
- prilohy.tex

chapters/ ..... Zdrojové .tex kapitol práce
- analysis.tex
- bibliography.bib
- evaluation.tex
- introduction.tex
- summary.tex
- synthesis.tex

figures/ ..... Obrázky a schémy práce v .pdf formáte
- addcngtowin.pdf
- cngprimarch.pdf
- cryptoapi_arch.pdf
- dizajnrdinstrukcii.pdf
- os.pdf
- osinit.pdf
- rng sumar.pdf
- rngapi.pdf
- schema prístupu generátora.pdf
- vm.pdf

```
    └── zl.pdf
├── .gitignore
├── acronyms.tex ..... Zdrojový .tex zoznamu skratiek
├── compile_template.bat ..... Skript pre kompiláciu šablóny
├── glossary.tex
├── kithesis.cls
├── kithesis.pdf
├── kithesis.sty
├── kithesis.synctex(busy)
├── latexmkrc
├── README.md
├── thesis.pdf ..... Bakalárska práca v .pdf
└── thesis.tex..... Zdrojový .tex koreňu šablóny
```